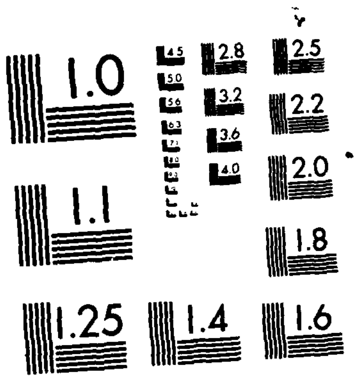
AD-A181 829 ARTIFICIAL INTELLIGENCE STUDY (AIS)(U) ARMY CONCEPTS ANALYSIS AGENCY BETHESDA MD R B MODJESKI FEB 87 CAA-RP-87-1 1/3

UNCLASSIFIED F/G 15/6 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RESEARCH PAPER
CAA-RP-87-1

ADF860111

ARTIFICIAL INTELLIGENCE STUDY (AIS)

FEBRUARY 1987

PREPARED BY
RESEARCH AND ANALYSIS SUPPORT DIRECTORATE
US ARMY CONCEPTS ANALYSIS AGENCY
8120 WOODMONT AVENUE
BETHESDA, MARYLAND 20814-2797

CAA

CAA-RP-87-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CAA-RP-87-1	2. GOVT ACCESSION NO. ADF860111	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Artificial Intelligence Study (AIS) (research paper)		5. TYPE OF REPORT & PERIOD COVERED Final 8/85 - 10/86 6. PERFORMING ORG. REPORT NUMBER CAA-RP-87-01
7. AUTHOR(s) Dr. Richard B. Modjeski		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS US Army Concepts Analysis Agency 8120 Woodmont Avenue Bethesda, MD 20814-2797		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE February 1987 13. NUMBER OF PAGES 216
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution Unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence; Combat Operations Research; Computer Architecture, Computerized Simulation; Computer Software; Expert Systems; Knowledge Representation, LISP; Studies and Analysis, Wargames

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper provides a preliminary overview of Artificial Intelligence (AI) software, hardware, and toolkits that may be useful for the U.S. Army Concepts Analysis Agency's (CAA's) computer simulation and study efforts. AI technologies were surveyed in relation to their applicability to CAA's study process and training programs. A glossary that defines AI and simulation techniques and a prototype expert system was also included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

THIS PAGE NOT USED

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

RESEARCH PAPER
CAA-RP-87-1

ARTIFICIAL INTELLIGENCE STUDY (AIS)

FEBRUARY 1987

PREPARED BY
RESEARCH AND ANALYSIS SUPPORT DIRECTORATE
US ARMY CONCEPTS ANALYSIS AGENCY
8120 WOODMONT AVENUE
BETHESDA, MARYLAND 20814-2797

DEPARTMENT OF THE ARMY
US ARMY CONCEPTS ANALYSIS AGENCY
8120 WOODMONT AVENUE
BETHESDA, MARYLAND 20814-2797

REPLY TO
ATTENTION OF

CSCA-RSR

15 MAY 1987

SUBJECT: Artificial Intelligence Research Paper

SEE DISTRIBUTION

1. The U.S. Army Concepts Analysis Agency (CAA) initiated an investigation of Artificial Intelligence (AI) technology and its possible applications to CAA's studies and analysis processes. The effort, which was started in September 1985, has yielded some observations that may be useful to other individuals outside CAA.

2. The research paper is being provided to you for your information. Questions or inquiries may be directed to the Chief, Advanced Research Projects Office (ARPO), U.S. Army Concepts Analysis Agency (CSCA-RSR), 8120 Woodmont Avenue, Bethesda, Maryland 20814-2797, (301) 295-0529 or AUTOVON 295-0529.

[illegible]

E. B. VANDIVER III
Director

DISTRIBUTION:
APPENDIX I

CAA	ARTIFICIAL INTELLIGENCE STUDY (AIS)	STUDY SUMMARY CAA-RP-87-1

THE REASON FOR PERFORMING THE STUDY was to gain a preliminary overview of adapting artificial intelligence (AI) and expert system technology to the theater-level for modeling and for the CAA study process.

THE PRINCIPAL ACCOMPLISHMENTS of the study are:

(1) AI software, hardware, and toolkits were surveyed for their applicability to CAA's study and analysis processes.

(2) AI techniques and methodologies were surveyed to determine their application to CAA study process.

(3) A small demonstration project was developed using the Command and Control (C^2) logic from CAA's FORCEM simulation model. The code was developed using the Knowledge Engineering Enviornment (KEE) toolkit on a Symbolics 3670 Lisp Machine.

THE MAIN ASSUMPTIONS OF THE STUDY are:

(1) AI/expert system technology may be applicable to military modeling.

(2) Rule-based solutions may be implemented to support computer simulation models used at CAA.

THE SPONSOR was the Director, US Army Concepts Analysis Agency.

THE STUDY EFFORT was directed by Dr. Richard B. Modjeski, Advanced Research Projects Office of the US Army Concepts Analysis Agency.

COMMENTS AND QUESTIONS may be addressed to the Director, US Army Concepts Analysis Agency, ATTN: CSCA-RSR, 8120 Woodmont Avenue, Bethesda, MD 20814-2797.

Tear-out copies of this synopsis are inside back cover.

CONTENTS

FIGURES

TABLES

ARTIFICIAL INTELLIGENCE (AI) STUDY

CHAPTER 1

INTRODUCTION

1-1. PURPOSE. The Director of the US Army Concepts Analysis Agency (CAA) tasked the Research Analysis Support Directorate (CSCA-RS) to study adapting artificial intelligence (AI) and expert system technology to theater-level force modeling. This study is a preliminary overview of AI technology and its application to the study and analysis process that is conducted at CAA.

1-2. BACKGROUND. CAA examines problems for the Army Staff (ARSTAF) and major Army commands (MACOMs). Analytic studies conducted by CAA assess capabilities, develop alternatives, or improve the comprehension of a sponsor's problem.

a. Many of CAA's analytic efforts use computer simulation models to develop a numerical representation of Army combat. The mathematical models are used to explore quantitative relationships about combat objects, such as tanks or terrain, and processes such as attrition, movement, or combat support (CS).

b. Hughes (1984) suggested the following nine categories of military models:

(1) Battle Planning. The preparation for wartime operations based on friendly and enemy orders of battle, existing strategic or tactical environment, and specific missions or objectives.

(2) Wartime Operations. The conduct of the war. Distinguished from battle planning by available current combat data and known, immediate military objectives.

(3) Weapon Procurement. The selection from among competing weapon systems or characteristics for procurement in the near future.

(4) Force Sizing. Deciding how many weapons of which types to (a) operate, (b) support, and (c) procure for the future, either in the defense establishment as a whole, or in a major component such as the Army or the nuclear weapons arsenal.

(5) Human Resource Planning. The design and operation of manpower, personnel, training, and assignment systems.

(6) Logistic Planning. The design and operation of all manner of military logistic support.

(7) **National Policy Analysis.** Supramilitary actions that influence, or are influenced by, military considerations such as arms treaties or subsidies of commercial transportation systems.

(8) **Command and Control, Communication, and Intelligence (C^3I).** May be embedded in any of the preceding seven categories of models but are often treated as a category by themselves.

(9) **Cost.** Another type of model which may be embedded in any of the above categories.

c. The computer simulation models at CAA have been developed using higher order languages. CAA currently operates over 30 major simulation systems using 120 computer models. Some of these models require more than 2 million words of memory, demand more than 12 hours of execution time, and need more than 100 separate executions per study. Some of the smaller stochastic models used at CAA often require more than 1,000 executions per study (Vandiver, 1985). CAA has grouped its models by function. The following five functional categories have been used to classify CAA models:

(1) Maintain readiness

(2) Direct force modernization

(3) Mobilize and deploy forces

(4) Conduct tactical warfare

(5) Sustain field operations

d. Recent artificial intelligence (AI) technology points to the inclusion of object-oriented programing and rule-based systems in computer simulation programs. Representing knowledge of specialized fields of expertise may aid in the analysis of complex simulation models. CAA's work in the development and analysis of models and model output may also be aided by AI technology. Portions of simulation models, such as command and control (C^2), may be enhanced by using expert system techniques.

e. AI methods may provide a means of developing systems of knowledge and inference strategies to simulate a military expert's behavior within a limited domain. For example, an expert advisory system could assist analysts in structuring the input vectors in the COSAGE Model.

1-3. STUDY PHASES. This study covers the period August 1985 through January 1987. Three development phases were completed. The principal goals of the study phases were:

a. Phase I - concept development

(1) AI hardware assessment plan

(2) Review of CAA's projects by study

(3) CAA's skill assessment

b. Phase II - proof of concept: AI prototype

c. Phase III - final development: Research Paper

1-4. STUDY OBJECTIVES. The AI study objectives were to:

a. Survey AI hardware, software, and tool kits that may be used to enhance CAA's models, methodologies, and studies.

b. Survey CAA's current projects by study, model, and/or methodologies.

c. Survey CAA's skills, knowledge, and abilities to determine what type of AI equipment and training are most appropriate to accomplish CAA's mission.

d. Propose ways to match AI technology with CAA models and studies to achieve near-term applications.

e. Develop a small demonstration project which applies AI technology to CAA material using tools which are currently available or will be acquired in the next 60 days from 1 August 1985.

1-5. ASSUMPTIONS. The assumptions of the study were:

a. AI/expert system technology may be applicable to military modeling.

b. Rule-based solutions may be implemented to support computer simulation models used at CAA.

CHAPTER 2

AN OVERVIEW OF AI TECHNOLOGY AND CAA MODELS, METHODOLOGIES, STUDIES, AND TRAINING

2-1. INTRODUCTION. Chapter 2 first provides a conceptual framework for consideration of Artificial Intelligence (AI) hardware, software, and toolkits relative to US Army Concepts Analysis Agency's (CAA's) models, methodologies, studies, and training. In the context of CAA studies performed during fiscal year 1985, the framework provides a baseline for judging the difficulty in transferring AI technology into the Agency. The survey is targeted primarily at ways to improve the building of models, the management of data, and the conduct of studies and analyses. Apart from several topical categorizations, the survey gives much less attention to the subjects to be modeled and studied than it does to the processes of modeling and studying. The survey does not cover all subjects presently associated with AI. For example, speech recognition, image processing, and robotics may have to be modeled and studied as candidates for battlefield application. They have been excluded from the survey on the grounds that they seem much less relevant to the issues of how to model and how to study. The chapter is subdivided into five sections. The first section surveys AI software, knowledge representation, knowledge bases, expert systems, and AI toolkits. The second section focuses on AI hardware. The third section surveys CAA's models, methodologies, and studies. Section IV analyzes CAA training. The chapter summary is contained in Section V.

Section I. ARTIFICIAL INTELLIGENCE SOFTWARE

2-2. AN OVERVIEW OF AI SOFTWARE

a. Introduction. AI software can be compared with conventional data driven software. Waterman (1986) suggested the following four categories:

Table 2-1. A Comparison of Traditional and AI Software

	Traditional software	AI software
Representation	Uses data	Uses knowledge
Programing techniques	Algorithms	Heuristics
Processes	Repetitive	Inferential
Data bases	Numeric	Symbolic

b. LISP. LISt Processing (LISP) software provides the intelligence for most AI systems. Pylyshyn, Bledsoe, Feigenbaum, Newell, Nilsson, Reddy, Rosenfeld, Winograd, and Winston (1980) have suggested that the fundamental technical issues in symbolic processes in computer science (i.e., representing knowledge, controlling search, and inference) can be understood in terms of software development. The fundamental assumption in AI software is that human thought can be automated (i.e., computed). Without software the most efficient AI hardware is merely the sum of its wires and gauges. LISP is the language of choice for most AI programs. Wah (1987) has suggested that conventional programing languages are inadequate for AI programing because of the complexity of programing AI features and the inefficient symbolic pattern processing. LISP on a LISP machine allows the language to be directly represented into the computer architecture.

c. LISP Development. LISP was developed during the 1950s by John McCarthy (1978). The full exploitation of LISP for symbolic computation required more flexible computer architectures than were available during the late 1950s. It was during the advent of AI architectures in computer hardware during the late 1970s that LISP and its dialects (Common LISP, INTERLISP, MACLISP, SCHEME, and ZETALISP) developed into the standard language for AI research in the United States. Barr and Feigenbaum (1982) have developed a detailed account of LISP and its applications in AI research and automatic programing. AI software developers have used LISP as the basis of simulation language that has been useful for military modeling.

d. PROLOG. PROLOG is an alternative language and methodology for AI software development. PROgraming Language for LOGic (PROLOG) was developed in France in the early 1970s (Clocksin & Mellish, 1984). An assumption underlying PROLOG is that AI is a type of applied logic. PROLOG uses the rules of inference, theorems, and proof that are derived from predicate calculus (applied logic). A PROLOG-like system was adopted as the starting point for Japan's fifth generation AI effort (Moto-oka, 1981).

2-3. LISP AND ITS DIALECTS

a. LISP Data Structures

(1) Very large-scale symbolic computation requires special hardware and software tools. These tools need to address the nonnumerical processing issues of knowledge representation, search control, and mechanisms for inference. Abelson and Sussman (1985) suggested that a computer provides a format for expressing ideas about methodology. LISP allows intellectual control of complexity by building programing abstractions through the use of data structures. LISP and its dialects provide a reasonable approach to these issues. LISP provides the programer the ability to code symbolic processes through the evaluation of lists. Galambos, Abelson, and Black (1982) have suggested that human symbolic processing (i.e., cognitive processes involved in using stored knowledge to understand and interpret new data and events) can be represented in LISP data structures. Although the program may not use the exact method used by a human would to solve a problem, it satisfices (i.e., is equally acceptable) for the goal. LISP machines are built around LISP and function as a hardware/software environment for the programer. LISP environments have no formal operating system. Instead, LISP utilities provide the programer with all the functionality of a conventional operating system plus a unique flexibility of a symbolic language. This total hardware/software environment allows the programer to use parts of the machine in his code that are not available nor as transparent as in general purpose computers.

(2) LISP data structures make symbolic representation straightforward. There are two basic elements: an atom (also called a symbol or name tag), and a list. LISP lists are made up of two of the five primitive functions (i.e., a CONS whose CDR is a list). The other three LISP primitive functions are the CAR, ATOM, and EQ. McCarthy, Abrahams, Edwards, Hart, and Levin (1985) differentiated LISP from other programing languages by the following characteristics:

> LISP differs from most programing languages in three important ways. The first way is in the nature of the data. In the LISP language, all data are in the form of symbolic expressions usually referred to as S-expressions. S-expressions are of indefinite length and have a branch tree type of structure, so that significant subexpressions can be readily isolated. In the LISP programing system, the bulk of available memory is used for storing S-expressions in the form of list structures. This type of memory organization frees the programer from the necessity of allocating storage for the different sections of his program. The second important part of the LISP language is the source language itself which specifies in what way the S-expressions are to be processed. This consists of recursive functions of S-expressions. Since the notation for the writing of recursive functions of S-expressions is itself outside the S-expressions notation, it will be called the meta language. These expressions will therefore be called M-expressions. Third, LISP can interpret and execute programs written in the form of S-expressions. Thus, like machine language, and unlike most other higher level languages, it can be used to generate programs for futher execution.
> (McCarthy, Abrahams, Edwards, Hart, & Levin, 1985, p 1)

(3) LISP objects may be represented by data typed pointers. The datatype of the object is determined from the pointer plus additional datatype bits which tag the pointer. Object-oriented style allows simplification of highly complex programing problems. Datatypes are flexible. This allows the programer to control the representation and storage of data. Memory space used by a data object is allowed to grow as the program is written. This is called dynamic allocation, a feature critical to the use of lists that change frequently. LISP allows the programer to take an unknown piece of tagged data and check its type. The flexibility of datatyping allows the programer to stuff procedural information into objects' property lists (a data structure usually in tabular form that is

associated with each symbol in LISP). All procedures can call each other. Cohen and Feigenbaum (1982) have called this feature dynamic scoping. This allows variable values to be passed down the control chain of a program regardless of where they appear in the program.

(4) LISP uses a list as a fundamental building block. Every LISP object is represented as a unique place in the LISP machine's memory. Stored addresses are called pointers. There is only one representation of a number object in the entire LISP machine. LISP lists are represented in the LISP machine as a succession of CONSes. The CONS are the backbone that hold the internal representation of the list in the LISP machine. Figure 2-1 illustrates a LISP CONS. CONS are composed of a CAR (Contents of Address portion of Register) and a CDR (Contents of Decrement portion of Register). Both CAR and CDR have grown to denote additional meaning beyond their historical acronym.

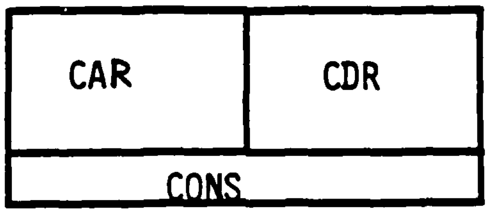


Figure 2-1. The LISP CONS

Gabriel (1985) has suggested that the advantage of determining the type of an object during runtime computation (and allowing program actions to take place that are associated with the type of object that was determined) may be offset by the burden of frequent type decoding on both the machine and the software.

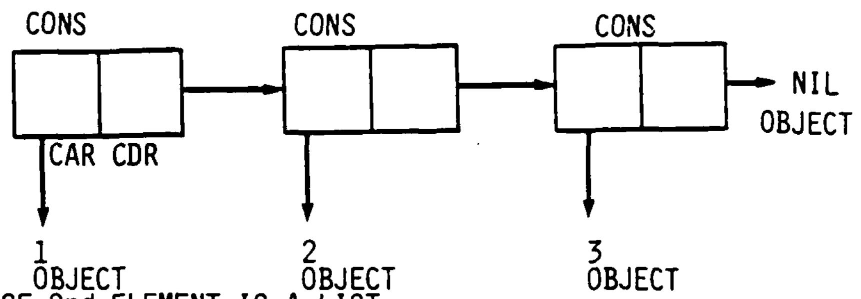
> There is a spectrum of methods for encoding the type of a LISP object and the following are the two extremes: the typing information can be encoded in the pointer, or it can be encoded in the object. If the type information is encoded in the pointer, then either the pointer is large enough to hold a machine address plus some tag bits (tagged architecture) or the address itself encodes the type. As an example, in the latter case, the memory can be partitioned

> into segments, and for each segment there is an entry in a master type table (indexed by segment number) describing the data type of the object in the segment. In MACLISP this is called the BIBOP scheme (Big Bag Of Pages) . . . in most LISPs, types are encoded in the pointer. However, if there are not enough bits to describe the subtype of an object in the pointer, the main type is encoded in the pointer, and the subtype is encoded in the object. For instance, in S-1 LISP, a fixed-pointer vector has the vector type in the tag portion of the pointer and the fixed-point subtype tag in the vector header. In SMALLTALK-80 and MDL, the type is in the object, not the pointer. In tagged architectures (such as the LISP Machine) the tags of arguments are automatically used to dispatch to the right routines by the microcode in generic arithmetic. In INTERLISP-D operations such as CAR compute the type for error checking purposes. In MACLISP, interpreted functions check types more often than compiled code where safety is sacrificed for speed. The speed of MACLISP numeric compiled code is due to the ability to avoid computing runtime types as much as possible. Microcode machines typically can arrange for the tag field to be easily or automatically extracted upon memory fetch. Stock hardware can either have byte instructions suitable for tag extraction or can arrange for other field extraction, relying on shift and/or mask instructions in the worst case. Runtime management of types is one of the main attractions of microcoded LISP machines.
>
> (Gabriel, 1985, pp 14-15)

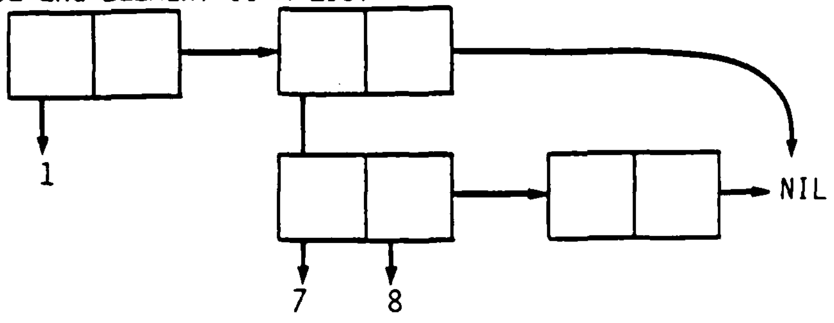
(5) The list structure provides a very flexible data structure. A list can represent symbolic information such as linked-list structures in the machine. The CONS can point to any object including another CONS cell by using its two-part structure to create a pathway to the location of the object in the computer's memory registers. CONS tie lists together like the vertebrae in a backbone. A list is not an object. LISP objects exist only inside the LISP machine. The user can only view a printed representative of a LISP object on the computer screen. Lists are CONS whose CDRs are a list. A LISP list may be a collection of LISP objects. Elements of a list are the CARS of each CONS. Therefore, since each CONS is composed of two objects (i.e., a CAR and CDR) the list (i.e., the CONS whose CDR is a list) cannot be a single object.

(6) Like a vertebrate backbone, CONS link LISP objects together. The CONS is equal to an element or a node of a list. The length of a LISP list may be determined by counting the number of CONS. A LISP list is a CONS whose CDR is a list. LISP lists end in NIL (i.e., the last CDR points to the LISP object NIL. Figure 2-2 illustrates LISP lists.

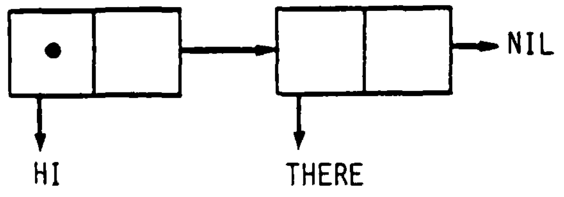
1. A LIST WITH ELEMENTS 1, 2, 3



2. A LIST WHOSE 2nd ELEMENT IS A LIST



3. A LIST (HI THERE)



4. A LIST ((HI THERE))

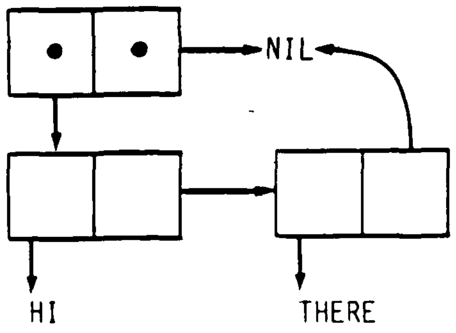


Figure 2-2. LISP Lists

(7) CONS may be linked by CDR coding. CDR coding reduces memory requirements for storing lists. The two-cell notion of the CONS uses the CAR, or first element, for symbol storage and the CDR, or second half, for list storage. The CDR is reduced to a two-bit word that can take only four values. Three of these values (CDR-NORMAL; CDR-NEXT; AND CDR-NIL) are used in list storage. The list starts with NORMAL and ends with NIL. CDR-NEXT and CDR-NIL represent lists as a vector. This is storage-efficient since CDR refers to a property of the memory, not its contents, and therefore only the CAR of the CONS is stored. CDR-ERROR, the fourth value, indicates address objects that are not in a list and are in error.

(8) CONS storage allows programs and data to be stored alike. Barr and Feigenbaum (1982) have suggested that the internal representation of LISP is its most important feature:

> LISP is unique among programing languages in storing its programs as structured data. This property is very important for several reasons. First, it is easy to write LISP programs that generate LISP expressions and programs, as in automatic-programing research. Second, functions can be passed as parameters . . . third, there is procedural representation of knowledge: LISP procedures for deducing facts can be stored in a database as if they were facts. Last and most important is the manner in which LISP can be a foundation for more advanced languages. The method is to write an interpreter, in LISP, for a new LISP-like language. Syntactic constructs of the new language are represented as multilevel lists just as in LISP itself, making the interpretation easy to do.
>
> (Barr & Feigenbaum, 1982, Vol II, pp 27-28)

The use of programs as data allows both the interactions with other programs and an audit trail where a program can explain how it made an inference (i.e., show its line of reasoning).

(9) Gabriel (1985) has suggested that both LISP machines and general purpose computers may use additional techniques for manipulating data structures beyond CAR/CDR and CONS:

> In addition to CONS cells, several implementations provide other basic data structures that are useful for building more complex objects. Vectors and vector-like objects (i.e., HUNKS in MACLISP) help build sequences and record structures; arrays build vectors (in implementations without vectors); matrices, and multidimensional records; and strings . . .. Further, many LISPs incorporate abstract data structuring facilities such as in the INTERLISP DATATYPE facility, the MACLISP EXTENDED DEFSTRUCT, and DEFVST facilities, and the LISP machine DEFSTRUCT and FLAVOR facilities. Several of these, especially the FLAVOR facility, also support Object Oriented Programing, much in the style of SMALLTALK.
>
> (Gabriel, 1985, p 12)

b. LISP Functions

(1) Functions do everything in LISP. Functions are the actors of the LISP world. LISP functions take a wide and possibly variable number and type of arguments and return a LISP object. LISP programs are function calls. Function calls are represented as lists.

(2) Functions provide the control structure for the LISP programing language. Functions take arguments, perform some operation, and return a LISP object. Functions can be called for effect or value. The application of functions to arguments guides the flow of a LISP program.

(3) Function definitions provide the steps of a LISP program. The assessment of arguments by functions guides the flow of control in LISP programs. Arguments may be symbols (atoms), lists, and functions. Large LISP programs are composed of many smaller LISP functions. The LISP systems that are used to write large programs have functions which edit, debug, trace, and evaluate functions.

(4) The evaluation function (called EVAL) is usually the largest utility function in the LISP world. EVAL is called implicitly in function calls. The function EVAL helps to run other functions and, in effect, run a LISP program by traversing the arguments given and returning a LISP object (Steele, 1984; Symbolics, 1985(b)). The function EVAL acts as an interpreter for LISP programs.

c. Dialects of LISP

(1) Most dialects of LISP are incompatible. LISP programs developed with one dialect often have hardware transportability problems that are associated with all large-scale development software. Touretzky (1984) views LISP programing as an aesthetic activity. His review of LISP dialects included five classes of LISP:

(a) MACLISP, COMMON LISP, and LISP MACHINE LISP

(b) FRANZ LISP

(c) UCI LISP and TLC-LISP

(d) INTERLISP

(e) P-LISP

(2) Barr and Feigenbaum (1982) have reviewed LISP programing language dialects and their applications. Some of the more experimental dialects of LISP such as QLISP, SAIL, POP-2, and FUZZY are reviewed. Common LISP, a hybrid LISP composed of MACLISP, SCHEME, SPICELISP, ZETALISP, STANDARD LISP, and NIL, was reviewed by Steele (1984). Common LISP has become the DOD standard LISP. The Defense Advanced Research Projects Agency (DARPA) has written its contracts for the Strategic Computing Initiative (SCI) to

use Common LISP as the development language and ADA as the production language. An overview from the Army's viewpoint can be found in the Army Soldier Support Institute Reference Book RB 18-155 (1984).

d. LISP Characteristics. Richardson (1983) summarized LISP-based software as having the following characteristics:

(1) Focus on symbol manipulation and list processing.
(2) Support of representations, which change dynamically.
(3) Support of flexible control by pattern matching rather than by procedure calls. (pattern matching is an abstract description of data symbols (i.e., rules) which partially specifies values that can be assumed where sets of patterns are matched against memory.)
(4) Supportive programing environments, including:
(a) An interactive (interpreted) language.
(b) A good editor (program construct oriented, not text oriented).
(c) Debugging facilities (traces, breaks).
(d) Standard systems input/output functions.

(Richardson (1983), p 42)

e. LISP Disadvantages. Barr and Feigenbaum (1982) note the following historical disadvantages of LISP. Many of the following disadvantages are no longer valid but provide a historical perspective on the development of LISP. The disadvantages are:

Ugly syntax. A common complaint about the list-structure format of LISP programs is that it makes them difficult to read. The only syntatic items are separators, such as spaces and parentheses, which provide most of the structure. This way of representing structure is convenient for the machine to read, by inconvenient for humans. In practice, facilities are provided for printing out programs so that the structure is also indicated with indention. No attempted alternative formats have caught on (with, perhaps, the exeception of the CLISP dialect provided in the INTERLISP system). The utility of LISP's structured representation of code outweights the nuisance of its external form.

One data type. Not having distinct data types in harmful when it prevents type-checking at run time, where bugs can often be detected early. However, many LISP systems do support additional data types, such as strings, arrays, and records (see also QLISP in Article VI.C2).

Inefficiency. Any language is relatively slow when executed interpretively. Speed is traded for convenience and extensibility. Some LISP compilers, however, produce fairly efficient code: Because of its use in the MACSYMA system, MACLISP produces very efficient code for numeric calclations.

Lack of a language standard. Unlike FORTRAN and other well-known languages, there has never been an attempt to agree on a standardized LISP. The absence of a language standard and the proliferation of incompatible versions make LISP badly suited to be a production language, and in AI research work there are severe difficulties in transporting LISP programs to machines running a different LISP.

(Barr & Feigenbaum, 1982, pp 28-29)

Common LISP has been developed under DOD/DARPA sponsorship by Steele (1984) to attempt a resolution of these disadvantages.

2-4. PROLOG (PROGRAMING FOR LOGIC)

a. Overview. The tradition of Western civilization has associated thought with logic. Many philosophers have entertained the notion that formal logic was similar to the thinking and reasoning processes associated with human cognitive behavior (i.e., thought processes). Systems of formal logic have evolved into a notion of how mankind and man ought to think (i.e., the optimization of thinking). Newell and Simon (1972) reviewed the historical development of programing digital computers to play chess at the Rand Corporation during the 1950s in order to perfect a symbolic simulation language to model air defense systems. This effort led to programing languages that used symbolic logic to prove theorems. Nilsson (1980) suggested that, in some limited areas of thinking (e.g., problem solving), artificial intelligence can be conceived of as a type of applied logic. This assumes that all knowledge can be reduced to logic statements. Programing for Logic (PROLOG) was created in France during the early 1970s in order to provide a machine efficient way of programing logic.

b. Predicate Logic. PROLOG is based on the assumption that logic statements can represent knowledge. Abelson and Sussman (1985) have suggested that PROLOG was a response to the notion that the control structure of a programing language can be combined with the operations of a logic management system. This bidirectional view of programing allows the goal of mathematical relations to yield multiple ill-defined outputs from a given set of inputs. Predicate calculus, a form of propositional logic, was chosen as the rule base for PROLOG's logical manipulation of symbols. It is also assumed that logic is the optimal vehicle for the manipulation of symbols. Kahane (1978) has suggested that predicate calculus is powerful enough to deal with a wide variety of atomic and compound sentences within the domain of logic. If this is true, then predicate calculus satisfies a necessary but not sufficient condition for representation of a wide variety of knowledge structures. Predicate calculus represents symbolic logic by use of symbolic predicates, variables, functions, and constants. Each symbol is called an object. Logical statements about objects are called predicates. Assertions of logic are restricted to horn clauses which connect two expressions by the logical term "or." Predicates indicate interrelationships between objects. For example, "SHOOT (TANK, GUN)" can be used to represent a domain of knowledge selected for logical analysis.

c. Logic Representation. Predicate symbols are used to represent the relationships of objects in a domain of knowledge used for logical analysis. The atomic formula "SHOOT (TANK, GUN)" can be combined with the atomic formula "MOVE (TANK, TURRET)" by logical connectives such as "AND," "OR," and "IMPLIES." Complex notions of objects and their relationships can be represented by using building blocks of atoms with connectives. Logical connectives may also represent the relationship within a formula (i.e., a horn clause) to create a hierarchy of objects. Formulas may also interrelate other formulas. These types of formulas are called conjunctions. At a higher level of complexity, a conjunction of formulas may describe a domain of knowledge called a world state (i.e., a world built of blocks of knowledge or a "blocks world"). Each of the component formulas must obey the rules of predicate calculus to yield a syntactically correct and logically legitimate expression called a well-formed formula, or WFF. World states or block worlds create a logical representation of a global database (i.e., knowledge base). Logical formulas provide a mechanism to chain backward through this global knowledge base in order to yield an inference. For example, a query like "Who is Major General Smith's commander?" would perform a deduction from a database of facts yielding an implication. The code would be expressed as follows:

```
( x) Commander-of-(Major General-Smith,x).

Commander (corps, General-Jones),

works-in (corps, Major General-Smith),

and

  Works-in (x,y)   Commander (x,z)   Commander-of (y,z)   ,
```

d. Task Specification. Programing in PROLOG involves the task of specifying objects. Facts about objects and rules about the relationships among objects must be specified in accordance with the rules of logic in order to program in PROLOG. The formulas, or horn clauses, provide a means of structuring a knowledge base. Clocksin and Mellish (1984) have suggested that PROLOG programs consist of specifications for sets of clauses. Clauses may be facts about objects or rules about how a problem solution may be inferred from facts. Formulas are used to enable the matching of one expression to a template expression (i.e., pattern matching). Unification algorithms go beyond pattern matching in using the rules of inference to prove theorems of quantified formulas that match subexpressions. Unification algorithms control backward chaining (i.e., search). Search may involve an algorithmic solution that may be recursive. Harmon and King (1985) summarized PROLOG programing in the following steps:

(1) Specify facts about objects and relationships.
(2) Specify rules about the objects and relationships.
(3) Ask questions using formal logic about objects and relationships.

(Harmon & King, 1985, p. 88)

e. **Summary.** Logic programing provides a series of programing techniques embodied within a language which uses the formal rules of logic to specify facts about objects and their relationships. These specifications of facts and relationships yield inferences. In domains where causal relationships are explicit, PROLOG provides a suitable programing environment. PROLOG allows programers to build programs by descriptions of objects, facts, and rules of relationship. Rules of formal logic provide an inference engine which yields conclusions and problem solutions. The inference engine is restricted to a backward chaining mechanism in PROLOG, thereby limiting the variety of ways to search a descriptive global knowledge base. PROLOG also allows automated reasoning with explanation of the conclusion. Slagle and Hamburger (1985) have used logic programing techniques to develop artillery resource allocation algorithms for assignment functions based on deduced target value relative to the total battle situation. Genesereth and Ginsberg (1985) have listed the following shortcomings of PROLOG:

(1) Its deductive methods are inadequate to reason analogically or with uncertain and missing data.

(2) It has no user-specified control of search and inference.

(3) It is difficult to debug.

The Budapest Institute for Coordination of Computer Techniques (SKZI) released a Modular PROLOG that facilitates programing and debugging PROLOG in 1976. Epstein (1986) has suggested that this version of PROLOG has been used for business and industrial applications in Eastern Europe. The Japanese have adopted PROLOG as the kernel language for their fifth generation AI effort (Moto-Oka, 1981). This is probably due to the ability of PROLOG to subsume the features of LISP more easily than the inverse of LISP subsuming the features of PROLOG (Yokoi, et al., 1982). The structure of PROLOG appears to be well suited for parallel processing architectures which have been declared as a major milestone in the Japanese fifth generation effort. The fifth generation computer is a very elegant series of machines which perform fast computation with numbers, symbols, and concepts. The computers will be built using advanced architectures with VLSI/VHSIC technology and gallium arsenide (GaAs) materials. In contrast, the first generation computer was characterized by vacuum tubes. The second generation of computer machinery was developed with transister. Increased speed in the third generation resulted from the incorporation of microcircuits into computer design while the fourth generation allowed this effort to continue with the use of silicon materials. Paragraph 2-11 develops the notion of AI hardware in greater detail.

2-5. AI SIMULATION LANGUAGES

a. Introduction. The use of LISP as an interpretive language can allow higher order languages to be written. Barr and Feigenbaum (1981) suggested the method to compose an interpreter, in LISP, for the target language. The syntactic constructs are literally written into the LISP language as hierarchical lists. Tables 2-2, 2-3, and 2-4 illustrate a LISP (and its utilities) interpreter (Cassels, 1985b, 1985c, and 1985d) for a standard SIMSCRIPT syntax (Johnson, 1984). Table 2-5 illustrates an application program for a job shop using the LISP interpreter (Cassels, 1985a).

Table 2-2. A SIMSCRIPT Interpreter Written in LISP
(Cassels, 1985d)
(page 1 of 3 pages)

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: SIMSCRIPT; Lowercase: Yes -*-


;;; This file contains the scheduler and other routines which make the
;;;   simulation work


;;; function used to suspend the current process

(defsubst suspend ()
  (stack-group-return :suspend))


;;; function used to activate a given process

(defsubst activate (process)
  (send *runnable-processes* :file process))


;;; function used to wait a given amount of time

(defmacro wait (amount &optional (time-unit :days))
  (let ((wait-time (ecase time-unit
                     ((:days :day :units :unit) amount)
                     ((:hours :hour) `(/ ,amount *hours*))
                     ((:minutes :minute) `(/ ,amount *hours* *minutes*)))))
    `(let ((wait-time ,wait-time))
       (when (> wait-time 0)
         (send *timer-heap* :insert *process* (+ *time* ,wait-time))
         (suspend)))))


;;; work is a synonym for wait

(deff work #'wait)


;;; convert time to days

(defun sim-time (days hours minutes)
  (declare (values days))
  (+ days (/ (+ hours (/ minutes *minutes*)) *hours*)))


;;; split a time into components, for pretty-printing

(defun time-parts (days)
  (declare (values days hours minutes))
  (multiple-value-bind (whole-days part-days)
      (floor days)
    (multiple-value-bind (whole-hours minutes)
        (floor (* part-days *hours*))
      (values whole-days whole-hours (* minutes *minutes*)))))
```

Table 2-2. A SIMSCRIPT Interpreter Written in LISP
(Cassels, 1985d)
(page 2 of 3 pages)

```
;;; the external event process

(define-process external-event-generator
               (external-events)
  (loop for (time function) in external-events
        do (wait (- (eval time) *time*))
           (eval function)))


;;; the scheduler

(defun start-simulation (external-events)
  (let ((*time* 0)
        (*timer-heap* (zl:make-heap))
        (*runnable-processes* (make-instance 'fifo-set))
        (*process* nil))

    (activate (make-external-event-generator :external-events external-events))

    (loop until (send *runnable-processes* :is-empty)

          do (loop until (send *runnable-processes* :is-empty)
                   do (setq *process* (send *runnable-processes* :remove-first))
                      (let ((status (send *process* :continue)))
                        (case status
                          (:abort (signal 'sys:abort)))))

             ;; after running all processes runnable at *time*, tick the clock
             (advance-clock))))


;;; the clock scheduler

(defun advance-clock ()
  (multiple-value-bind (ignore time non-empty)
      (send *timer-heap* :top)

    (when non-empty
      (setq *time* time)

      ;; activate all processes for this time
      (loop with (process start-time process-p)
            do (multiple-value-setq (process start-time process-p)
                 (send *timer-heap* :top))
            while (and process-p
                       (= start-time time))
            do (send *timer-heap* :remove)
               (activate process)))))
```

Table 2-2. A SIMSCRIPT Interpreter Written in LISP (Cassels, 1985d) (page 3 of 3 pages)

```
;;; what a resource does when you request it

(defmethod (resource :request) (amount priority)
  (ignore priority)                          ; ---- for now
  (cond ((≥ number-of-units amount)
         (decf number-of-units amount))
        (t
         (send *process* :set-waiting-for-resource self)
         (send *process* :set-waiting-for-units amount)
         (send processes-waiting :file *process*)
         (suspend)
         (send *process* :set-waiting-for-units 0))))


;;; what a resource does when you relinquish it

(defmethod (resource :relinquish) (amount)
  (incf number-of-units amount)
  (loop until (send processes-waiting :is-empty)
        as process = (send processes-waiting :first)
        as request-amount = (send process :waiting-for-units)
        while (≥ number-of-units (send process :waiting-for-units))
        do (send processes-waiting :remove-first)
           (activate process)
           (decf number-of-units request-amount)))


;;; function used to request a resource

(defsubst request (resource units &optional (priority 0))
  (send resource :request units priority))


;;; function used to relinquish a resource

(defsubst relinquish (resource units)
  (send resource :relinquish units))
```

Table 2-3. SIMSCRIPT Definitions Written in LISP
(Cassels, 1985c)
(page 1 of 3 pages)

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: SIMSCRIPT; Lowercase: Yes -*-


;;; This file contains data structure definitions used by the Simscript system


;;; definitions of Simscript sets

(defflavor set
        ((set nil))
        ())

(defmethod (set :size) ()
  (length set))

(defmethod (set :is-empty) ()
  (null set))

(defmethod (set :is-in-set) (item)
  (member item set))

(defmethod (set :first) ()
  (when (null set)
    (ferror "The set is empty"))
  (first set))

(defmethod (set :remove-first) ()
  (when (null set)
    (ferror "The set is empty"))
  (pop set))

(defmethod (set :remove) (item)
  (let ((new-set (delete item set)))
    (when (eq new-set set)
      (ferror "The item isn't in the set"))
    (setf set new-set)))

(defflavor lifo-set
        ()
        (set))

(defmethod (lifo-set :file) (item)
  (push item set))

(defflavor fifo-set
        (set-end)
        (set))

(defmethod (fifo-set :after :init) (ignore)
  (setf set-end (locf set)))

(defmethod (fifo-set :file) (item)
  (let ((new-set-end (ncons item)))
    (setf (cdr set-end) new-set-end)
    (setf set-end new-set-end)))

(defmethod (fifo-set :after :remove-first) ()
  (when (null set)
    (setf set-end (locf set))))

(defmethod (fifo-set :after :remove) (item)
  (cond ((null set)
         (setf set-end (locf set)))
        ((eql item (car set-end))
         (setf set-end (last set)))))

(compile-flavor-methods lifo-set fifo-set)
```

Table 2-3. SIMSCRIPT Definitions Written in LISP
(Cassels, 1985c)
(page 2 of 3 pages)

```
(defflavor average-tallier
        ((total 0)
         (number 0))
        ()
  (:gettable-instance-variables total number))

(defmethod (average-tallier :average) ()
  (if (zerop number) 0
      (/ total number)))

(defmethod (average-tallier :reset) ()
  (setf total 0)
  (setf number 0))

(defmethod (average-tallier :new-value) (value)
  (incf total value)
  (incf number 1))

(compile-flavor-methods average-tallier)


;;; variables used by the Simscript system

(defvar *process*)                          ; currently-running process
(defvar *timer-heap*)
(defvar *runnable-processes*)               ; FIFO set of activated processes

(defvar *time*)                             ; simulated time, in days
(defvar *hours* 24)                         ; number of simulated hours in a day
(defvar *minutes* 60)                       ; number of simulated minutes in an hour


(defun find-variables (variable-initialization-list)
  (mapcar #'(lambda (variable-initialization)
              (etypecase variable-initialization
                (cons (first variable-initialization))
                (symbol variable-initialization)))
          variable-initialization-list))

(defflavor resource
        ((number-of-units 0)
         (processes-waiting (make-instance 'fifo-set))
         (processes-using nil))
        ()
  (:gettable-instance-variables number-of-units processes-waiting processes-using)
  (:initable-instance-variables number-of-units))

(defmacro define-resource (name attributes)
  (let ((attribute-names (find-variables attributes)))
    `(progn (defflavor ,name ,attributes
                   (resource)
              (:initable-instance-variables ,@attribute-names)
              (:gettable-instance-variables ,@attribute-names))

            (defmacro ,(intern (string-append "MAKE-" name)) (&rest args)
              (declare (arglist &key ,@attribute-names number-of-units))
              `(make-instance ',',name ,@args))

            (compile-flavor-methods ,name))))
```

Table 2-3. SIMSCRIPT Definitions Written in LISP
(Cassels, 1985c)
(page 3 of 3 pages)

```
;;; this is how the Simscript system represents processes

(defflavor process
        ((stack-group (make-stack-group "Simscript Process"
                                         :regular-pdl-size #o2200
                                         :special-pdl-size 200
                                         ))
         (waiting-for-resource nil)
         (waiting-for-units 0))
        ()
  (:settable-instance-variables waiting-for-resource waiting-for-units))

(defmethod (process :after :init) (ignore)
  (stack-group-preset stack-group 'send self :process-stuff)
  (forward-stack-group-bindings stack-group))

(defun forward-stack-group-bindings (stack-group)
  (let ((binding-stack-top sys:%binding-stack-pointer)
        (binding-stack-low sys:%binding-stack-low))
    (setf (sys:sg-binding-stack-pointer stack-group)
          (sys:%make-pointer-offset sys:dtp-locative
                                    (sys:sg-binding-stack-low stack-group)
                                    (sys:%pointer-difference binding-stack-top
                                                             binding-stack-low)))
    (loop for my-pointer = binding-stack-low
                         then (sys:%make-pointer-offset sys:dtp-locative my-pointer 2)
          for his-pointer = (sys:sg-binding-stack-low stack-group)
                          then (sys:%make-pointer-offset sys:dtp-locative his-pointer 2)
          until (sys:%pointer-lessp binding-stack-top my-pointer)
          ;; forward his binding stack to mine
          do (sys:%p-store-tag-and-pointer his-pointer
                                           sys:dtp-external-value-cell-pointer
                                           my-pointer)
             (setf (location-contents (sys:%make-pointer-offset sys:dtp-locative
                                                                 his-pointer 1))
                   (location-contents (sys:%make-pointer-offset sys:dtp-locative
                                                                 my-pointer 1))))))

(defmethod (process :process-stuff) ()
  (condition-case ()
      (send self :process)
    (sys:abort (stack-group-return :abort)))
  (stack-group-return :done))

(defmethod (process :continue) ()
  (funcall stack-group nil))

(defmacro define-process (name attributes &body body)
  (let ((attribute-names (find-variables attributes)))
    `(progn (defflavor ,name ,attributes
                  (process)
              (:initable-instance-variables ,@attribute-names))

            (defmethod (,name :process) ()
              ,@body)

            (defmacro ,(intern (string-append "MAKE-" name)) (&rest args)
              (declare (arglist &key ,@attribute-names))
              `(make-instance ',',name ,@args))

            (compile-flavor-methods ,name))))
```

Table 2-4. SIMSCRIPT System Declaration Utilities in a LISP Program (Cassels, 1985b)

```
;;; -*- Mode: LISP; Syntax: Zetalisp; Package: USER; Lowercase: Yes -*-


(defpackage simscript
  (:use symbolics-common-lisp common-lisp-global)
  (:colon-mode :external)
  (:export *process*
           *time*
           *hours*
           *minutes*
           fifo-set
           lifo-set
           average-tallier
           define-resource define-process
           request relinquish activate work wait
           sim-time
           start-simulation))


(defpackage simscript-user
  (:use symbolics-common-lisp common-lisp-global simscript)
  (:nicknames sm-user))


(defsystem Simscript
  (:name "Prototype Simscript")
  (:pathname-default "Stony-Brook:>Cassels>Simscript>")
  (:package simscript)

  (:module definitions ("Simscript-definitions"))
  (:module code ("Simscript-system"))

  (:compile-load definitions)
  (:compile-load code (:fasload definitions)))
```

Table 2-5. A SIMSCRIPT Job Shop Program Written in LISP
(Cassels, 1985a)
(page 1 of 2 pages)

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: SIMSCRIPT-USER; Lowercase: Yes -*-


;;; Preamble


(defvar *production-centers*)

(defvar *cycle-time-tallier*)

(defvar *last-report-date*)


(defstruct task
  doer
  duration)


(define-resource production-center
        (machine-type))


(define-process job
                (arrival-time
                 (routing-set (make-instance 'fifo-set)))
  (loop for task in routing-set
        do (request (task-doer task) 1)
           (work (task-duration task) :hours)
           (relinquish (task-doer task) 1))
  (send *cycle-time-tallier* :new-value (- *time* arrival-time)))


(defun main (machine-allocation external-events)
  (let ((*production-centers*
          (loop for (machine-type number-of-units) in machine-allocation
                collect (make-production-center :machine-type machine-type
                                                :number-of-units number-of-units)))
        (*last-report-date* 0)
        (*cycle-time-tallier* (make-instance 'average-tallier)))
    (start-simulation external-events)))

(defun jobinit (tasks)
  (let ((routing-set
          (loop for (machine duration) in tasks
                collect (make-task :doer (find machine
                                               *production-centers*
                                               :test #'(lambda (m p)
                                                         (eql m (send p :machine-type)))
                                   :duration duration))))
    (activate (make-job :arrival-time *time*
                        :routing-set routing-set))))

(defun analysis ()
  (format t "~5&   E X A M P L E   J O B   S H O P   S I M U L A T I O N")
  (format t "~3&REPORTING PERIOD        ~5,1F HRS.  TO  ~5,1F HRS."
          (* *last-report-date* *hours*)
          (* *time* *hours*))
  (format t "~2&JOBS COMPLETED DURING PERIOD :        ~3D"
          (send *cycle-time-tallier* :number))
  (format t "~&AVERAGE COMPLETION TIME :             ~4,1F HRS."
          (* (send *cycle-time-tallier* :average) *hours*))
  (setf *last-report-date* *time*)
  (send *cycle-time-tallier* :reset))
```

Table 2-5. A SIMSCRIPT Job Shop Program Written in LISP (Cassels, 1985a) (page 2 of 2 pages)

```
(main '((forge 1) (saw 1) (press 2)
        (lathe 3) (mill 1) (shaper 1)
        (grinder 1) (welder 1) (drill 2)
        (finish 1))
      '(((sim-time 0 0 00)
         (jobinit '((saw 2.0) (lathe 10.00)
                    (grinder 2.00) (shaper 4.00) (finish 2.00))))
        ((sim-time 0 0 30)
         (jobinit '((forge 1.00) (shaper 1.00)
                    (lathe 2.00) (shaper 5.00) (mill 6.00))))
        ((sim-time 0 2 30)
         (jobinit '((saw 1.00) (welder 1.00)
                    (drill 2.00) (lathe 4.00) (mill 2.00))))
        ((sim-time 0 2 45)
         (jobinit '((drill 0.50) (welder 2.00)
                    (press 1.00) (welder 1.00) (finish 2.00))))
        ((sim-time 0 4 00)
         (analysis))
        ((sim-time 0 4 30)
         (jobinit '((mill 10.0) (lathe 2.00) (shaper 4.00))))
        ((sim-time 0 8 00)
         (analysis))
        ((sim-time 1 0 0)
         (analysis))
        ((sim-time 2 0 0)
         (analysis))))
```

```
     E X A M P L E   J O B   S H O P   S I M U L A T I O N

REPORTING PERIOD            0.0 HRS.  TO    4.0 HRS.

JOBS COMPLETED DURING PERIOD :          0
AVERAGE COMPLETION TIME :                 0.0 HRS.


     E X A M P L E   J O B   S H O P   S I M U L A T I O N

REPORTING PERIOD            4.0 HRS.  TO    8.0 HRS.

JOBS COMPLETED DURING PERIOD :          0
AVERAGE COMPLETION TIME :                 0.0 HRS.


     E X A M P L E   J O B   S H O P   S I M U L A T I O N

REPORTING PERIOD            8.0 HRS.  TO   24.0 HRS.

JOBS COMPLETED DURING PERIOD :          5
AVERAGE COMPLETION TIME :               16.8 HRS.


     E X A M P L E   J O B   S H O P   S I M U L A T I O N

REPORTING PERIOD           24.0 HRS.  TO   48.0 HRS.

JOBS COMPLETED DURING PERIOD :          0
AVERAGE COMPLETION TIME :                 0.0 HRS.
```

b. Object-Oriented Programing for AI Simulation

(1) Introduction. Two unique contributions from AI software to computer science which overlap in their development are the notion of objects (Stefik and Bobrow, 1986) and the mathematics of object inheritance with program and data (Touretzky, 1986). Both concepts are used in object-oriented programing. Object-oriented program methodology may be distinguished from the object orientation of LISP as a pointer system (i.e., ZETALISP). Object-oriented programing is the convergence of many languages. The programing features of these languages form a class of programing techniques which may be useful for simulation and modeling if standardized. The programing techniques use objects as entities (i.e., who are attributes of data and procedures) to send messages that affect other objects. The objects may be composed of other objects and instantiated with a pattern of variables to model complex objects and their processes in the real world. The simulation of these objects with message passing allows the simulation of effects among data and procedures represented by these objects. Qualitative simulation about the physical world (Bobrow, 1985; Kuipers, 1986) has proven useful for planning and understanding the dynamics and kinematics of complex real world problems in physics and chemistry. Bellman (1978) has demonstrated that many AI techniques can be applied to operations research and decisionmaking problems. Simon and Newell (1958) suggested that heuristic problem-solving techniques like object-oriented programing may allow operations research to structure a better class of problem for management decisionmakers. Object-oriented programing techniques may allow greater fidelity of simulation at the same or lower investment cost in programer time and effort. The payoff in combat simulation may be enormous.

(2) Generic Objects. Generic objects can be used to model real-world objects at a very high level of abstraction. Object-oriented programing uses the notion of objects to create global structures that can be specified by a wide variety of attributes. Generic objects may be created as programer-defined data types. Some of the attributes of objects may be unknown. These generic objects may produce objects that are instantiated with some values associated with the generic object but not other values. This allows a programer to build specifications at a later time or omit specifications from some objects in a set of objects. This level of abstraction also allows programers to build simulations without complete specification of structure or function.

(3) Objects, Instances, and Inheritance. Generic objects may produce other objects instantiated with specific values from some or all of the slots in the generic object. Networks of these related objects also allow lower order objects to inherit attributes of one or more higher order objects (multiple inheritance). The features of multiple inheritance and instantiation allow objects to maintain a functional relationship with other sets of objects while maintaining a unique identity.

(4) Message Passing. Messages are used to simulate complex interactions among objects. This enables a programer to structure the design of a simulation while it is occurring. Messages can be used to specify operations to be performed (i.e., procedural calling). This enables objects to have simulated behaviors that can be combined with structural relationships. Similar objects can interpret operations to be performed in a similar manner while different objects can interpret the same message to perform a different operation. Stefik and Bobrow (1986) suggested that the data abstraction feature of message passing is unique to the object-oriented class of programing languages. Message passing among hierarchies of highly related objects appear to provide a perfect tool for modeling the interaction between "things" and "processes" in complex and ill-defined problems (i.e., systems). This notion was also supported by Dockery and van den Driessche (1984) in the event-stepped simulation of command and control. Waterman (1986) has also suggested that message passing can allow new options in simulation such as concurrent and asynchronous simulation:

> Like rules, objects and message passing provide a natural way for handling processes where the wealth and variety of data configurations determine reasoning paths. Objects and message passing also provide a way to specify concurrent, asynchronous operations; it's possible to simulate many unrelated processes occurring at the same time. For example, in SWIRL (Simulating Warfare in the ROSS Language), command centers can send locate-and-destroy messages to air force bases while ground radars are communicating with filter centers about the latest targets detected.
>
> (Waterman, 1986, pp 98-99)

c. **Rand's AI Simulation Languages.** Three language were built by the Rand Corporation for simulation. Each language uses more sophisticated AI techniques than its predecessor. The three languages--ROSIE, ROSS, and ABEL--have reflected the hardware and software technology available at the time of their development. Klahr and Waterman (1986) give an overview of AI research at RAND which includes the background of these three languages.

(1) **ROSIE.** Many of the conventions useful to simulation in another AI language called SMALLTALK and a simulation language called SIMULA were incorporated into the first Rand simulation language called Rule Oriented System for Implementing Expertise (ROSIE). Waterman and Hayes-Roth (1982) evaluated ROSIE and found that the English-like code of the language provides a natural link between the programer and the user. Data and assertion (rules) are stored in a relational (global) data base. Fain, et al. (1985) designed the rule-oriented data base to be structured by a user-developed interpreter. Computer simulation models written in ROSIE were found difficult to modify, and the language has not been used in recent years.

(2) **ROSS.** An object-oriented simulation language called Rule Oriented Simulation System (ROSS) (McArthur & Klahr, 1982) was developed at the Rand Corporation to simulate behavioral analysis of real-time situations. Many complex simulations were developed with ROSS that analyzed military command and control behavior and combat decisionmaking. The object-oriented,

message-passing format allowed the analysis of indirect effects of an object's interaction with a whole system. Complexity in simulation can be simplified by instantiation of a generic object with a myriad of attribute values. ROSS was used to build two battle management simulations. Simulating Warfare in the ROSS Language (SWIRL) (Klahr, et al., 1982) was used to simulate an air penetration by Red forces attacking a Blue defensive area. Tactical Warfare in ROSS Language (TWIRL) (Klahr, et al., 1984) used ROSS to simulate a hasty river crossing from the march. In this simulation TWIRL provided a testbed for analysts to design strategy for C^3 electronic countermeasures, electronic warfare, and electronic combat. Nugent and Wong (1984) used ROSS in a unique and exciting way. The concept uses a notion developed earlier by Bonasso (1981) of the Battlefield Environment Model (BEM) which simulated Red unit signatures that were tracked on the battlefield by Blue (NATO) sensors. This information was fed to an expert system (called the ANALYST) to determine battlefield tactical deployment. The threat event generator, target data base, and sensor models used ROSS software on a VAX 11-780 hardware configuration. The ANALYST program received the VAX output while running on a Symbolics 3670 AI machine. The ANALYST was used in a DARPA AirLand Battle Management (ALBM) technology insertion experiment by Zymlemen (1986). The ANALYST software with a LISP machine was placed into a C^2 operational environment of an Army infantry division. The graphics, windowing (i.e., mouse sensitive menus), and inference explanation features of the ANALYST were readily accepted by all levels of the command. Dockery (1984) and Dockery and Van den Driessche (1985) also used ROSS to develop a simulation which demonstrated cause and effect for C^2. Gunsch and Hebert (1983) used ROSS to write a generic planning task simulator for a HAWK missile battalion command to use when choosing emplacement sites for firing batteries.

(3) ABEL. ABEL was developed as a preprocessor for the C language by LaCasse (1984). Shapiro, Hall, Anderson, and LaCasse (1985) suggested that ABEL evolved from a hand-translation of ROSIE into a simpler "pseudo-ROSIE" and the development of an automatic translator to convert the pseudo-ROSIE to C language. ABEL is not interpretive and generally runs 500 to 1,000 times faster than ROSIE according to Davis (1984). ABEL was used to program AI techniques such as production rules, goal-directed search, and pattern matching algorithms into the Rand Strategy Assessment Center (RSAC). The global model combines wargaming with analytic simulation by allowing gamers to input gamer logic at various levels in the simulation or by providing the program logic by default. ABEL is an AI-like simulation language designed for production runs of the RSAC model.

g. Criteria for AI Simulation Languages. The general criteria for a good AI simulation language would be similar to the same criteria for the LISP language outlined by McCarthy (1978):

(1) Computing with symbolic expressions rather than numbers; that is, bit patterns in a computer's memory and registers can stand for arbitrary symbols, not just those of arithmetic.

(2) LISP processing that is representing data as linked-list structures in the machine and multilevel lists on paper.

(3) Control structures based on the composition of functions to form more complex functions.

(4) Recursion as a way to describe processes and problems.

(McCarthy, 1978, pp 219-220)

h. Army Applications. A study by the Army Science Board Ad Hoc Subgroup on Artificial Intelligence and Robotics (Army Science Board, 1982) suggested that the Army Model Improvement Program (AMIP) be provided with the means to incorporate the new operational concepts made feasible by AI. The principal problem to implementation of AI concepts was found to be slowness in adapting technology-driven changes into existing programs before the changes were made obsolete by future technology and future threats.

2-6. KNOWLEDGE REPRESENTATION AND AI KNOWLEDGE BASES

a. Overview. The knowledge base is a computer data memory that is executable and provides power to the inference engine (i.e., interpreter) in an expert system software. Knowledge base programing methodologies map knowledge used by experts to solve problems. Knowledge bases contain both declarative (i.e., what to know) and procedural (i.e., what to do) types of knowledge. A favorite maxim of knowledge engineers in the AI community is "the power is in the knowledge (i.e., knowledge base)." A knowledge base can be defined as a global data base consisting of rules, facts, objects, and values. The assumption underlying a knowledge base is that it can sufficiently emulate human knowledge by using these facts, objects, rules, and values as data. Generally, the data in a knowledge base is equivalent to a data structure where pieces of knowledge are contained within that data structure. A notion of a knowledge base is for data to describe facts about an object, while knowledge is used to describe relationships among sets of objects. Knowledge representation can also be thought of as a multilevel notion of elements similar to that used in modeling, simulation, and analysis. Knowledge needed in building the model, specifying the input, modifying the model, and running the study can be conceived of as separate domains of knowledge interconnected at higher levels (i.e., knowledge interconnected by metaknowledge). The development of knowledge bases is a methodology that is only beginning to exist. Knowledge-based programs used by expert system software for industrial applications is about 3 years old. Methodologies for knowledge-based systems of any large size (i.e., greater than 2,000 rules) do not exist at the present time. Current knowledge systems are also limited at the present time to storage of human expertise in computer memory. Users can extract information from a knowledge base by using other software (e.g., an inference engine). Knowledge bases are often used in conjunction with inference engines and other software utilities (i.e., user interface and explanation software) to yield a total system that emulates a human expert. When configured properly, the system will use its knowledge base to assist the user in formulating more precise questions for performing problem-solving operations. Lesgold (1983) has suggested that representing expertise involves programing and refining discipline-specific knowledge and domain-specific strategies for use of that specialized knowledge. A characteristic of expert use of knowledge is flexibility in problem solving. Knowledge representation in human expertise is developed by increasing the amount of knowledge and practice with the knowledge base (i.e., the growth of judgmental reasoning). This representation methodology is not possible with current computer software systems. Expertise must be captured at its peak performance in the human expert and mapped into machine memory. The developmental aspect of human expertise is lost to the software system. The development of self-modifying systems will probably require a different methodology than that of attempting to capture human cognitive processes in machine memory. Symbolic representation of complex procedures of human knowledge must wait for more sophisticated software and faster multiprocessor hardware (i.e., single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) architectures) which emulate neurophysiologic processes. The following subparagraph reviews the current state of knowledge representation methodologies applicable to CAA and its mission. Some of the issues for military modeling and simulation are also addressed.

b. Knowledge Representation Methodologies. Knowledge representation involves programing a combination of data structures and interpretive procedures in a computer program. List-and-pointer data structure languages (i.e., LISP/PROLOG) can be used to represent facts and heuristics (i.e., rules of thumb used by experts). Complex AI software systems such as expert systems assume that most knowledge within a domain of human expertise can be structured into computer memory. Garzia, Garzia and Zeigler (1986) have suggested that the mixing of knowledge representation techniques with traditional simulation models results in a hybrid knowledge-based simulation system:

> The synergy between discrete-event simulation and the approaches that programmers take to develop artifical-intelligence software took some time to become apparent, but it is not being extensively exploited. AI knowledge-representation schemes are being incorporated into simulation models, resulting in so-called knowledge-based simulation systems. Such schemes can be used to organize information about the nature of the objects involved in simulation, speeding the development of new models. Knowledge systems can be embedded in the simulated system to model intelligent agents. A research group at Rand Corp. has been developing a simulation environment for war games in which humans can play against simulated intelligent opponents that model varying strategies and temperments. AI techniques also have great potential to improve modeling

> and simulation methodology. Expert systems encoding knowledge about simulation techniques may lessen the need for modelers to be experts in simulation programming, as well as advising on the selection of models or their components for specific purposes, and interpreting simulation results with expert statistical judgment. But progress in this area is not likely to be rapid.
> (Garzia, Garzia, & Zeigler, 1986, p 35)

The most useful methodologies currently used for knowledge representation involve rules, objects, active values, and logic.

(1) Rules. Rule-based systems assume that most human knowledge can be represented by "IF-THEN" rules and procedures. Newell and Simon (1972) suggested that most types of intelligent processes involved in human problem solving can be represented with fidelity and detail by "IF-THEN" rules. Rule-based or production systems use descriptions of symbols (i.e, rules) to characterize relevant situations and corresponding actions. Rules allow computer programs to compute changing data without a structured flow of control in the program. This method is useful when the data cause a lot of branching to occur during execution. When the "IF" in the rule is satisfied with facts, the "THEN" portion of the rule fires and causes some action to be performed. Hayes-Roth (1985) has characterized all rule-based knowledge representations by the properties that they:

- Incorporate practical human knowledge in conditional IF-THEN rules.
- Increase skill at a rate proportional to the enlargement of the knowledge base.
- Solve a wide range of possibly complex problems by selecting relevant rules and then combining the results in appropriate ways.
- Adaptively determine the best sequence of rules to execute.
- Explain their conclusion by retracing their actual lines of reasoning and translating the logic of each rule employed into natural language.

(Hayes-Roth, 1985, p 921)

Rule-based knowledge representation methodology partitions both the problem and the solution into understandable modules. The knowledge base (i.e., data memory) consists of rule data as symbolic expressions and facts as assertion about properties of the rules. The knowledge base is processed by an inference engine which computes the rule and facts against a working memory of temporary assertions to yield problem solutions or inferences with an explanation. The knowledge representation with rules is very restrictive to the problem domain. Attempts at a general problem solver (Newell and Simon, 1972) were unsuccessful. Problems are restricted to a narrow domain which must be specified in advance by the end-user and the expert.

(2) **Objects.** The notion of an object as a way to represent knowledge borrows heavily from the work on concept behavior in cognitive psychology. Concepts are any describable regularity of real or imagined events or objects. Characteristics which specify a concept are called defining attributes of the concept, and there are many rules and principles for describing binary partitions and relationships within and among concepts. Objects are similarly defined. Bobrow and Winograd (1985) suggested that an elegant organization for declarative knowledge is to focus it on a set of conceptual entities with attached descriptions such as objects. The notion of an object as being represented only by a description and its ability to inherit procedural as well as declarative properties is useful to capture the underlying complexity of data and knowledge. Stefik and Bobrow (1986) defined objects as:

> . . . entities that combine the properties of procedures and data since they perform computations and save local state. Uniform use of objects contrasts with the use of separate procedures and data in conventional programming.
>
> (Stefik & Bobrow, 1985, p 41)

This definition allows objects to function as both a knowledge representation methodology and a programing style. The object-oriented programing style is strong and effective when used as an add-on to LISP. The layering and abstraction of objects used to represent pieces of human memory is a powerful knowledge representation methodology. Object-oriented methodology represents instructions and data with objects. Hierarchies of objects can inherit properties from one or several objects (i.e., single and multiple inheritance). Objects can be related in the computer knowledge base by inheritance sets and message passing protocols. Within the class of objects, knowledge representation methodologies are the frame. Frames are used to represent objects. The notion of frames was developed by Minsky (1975, 1985), Bobrow, and Winograd (1977, 1985). Frames are data structures that internally define the interrelationships between embedded bits of declarative and procedural knowledge. Procedures within a frame can be attached to attribute descriptors of objects called slots. Slots can be used to activate the inference process of entire software systems. Slots may store values, sets of rules, attributes of objects, and procedural information. Slots may be used to attach complete or partial attributes to objects. Procedural information written in another language (i.e., LISP or PROLOG) can be attached to a frame. Frames also provide a global partitioning device for sets of production rules. This feature may serve as a management system for rule sets contained within the knowledge base.

(3) **Access-Oriented Knowledge Representation.** The structure of an access-oriented knowledge base is linked to using active values in a nested hierarchy with software demons. An active value is a procedure that is called when program data are read into the knowledge base or are changed. The concept of an active value was developed from the AI hardware notions of bit-mapped graphics (i.e., a computer screen display where each pixel image is tied to a memory cell) and bit boundary block transfer (bitblt)

(for performing Boolean operations graphically, such as dragging graphical parts from memory onto a computer screen and across the computer screen with a command or mouse click). Stefik, Bobrow, Mittal, and Conway (1983) defined active values as structures which have procedures that are invoked when variables are called. They are like electrical circuit probes that can be placed at various points in a circuit without changing the value of the circuit. In software the active value converts a variable reference into a procedural call. The concept of an access-oriented knowledge base is tied to a graphic superobject that aggregates active objects in a simulated environment. Alan Kay developed the notion of access-oriented knowledge base methodology during the 1970s at XEROX PARC (Perry & Wallick, 1985). Kay (1984) provided a detailed explanation of access-oriented knowledge bases in electronic spreadsheet application programs. In more complex software, a demon may be added to probe the values of variables and trigger computation or drive graphic images. An access-oriented knowledge base is composed of nested active values (i.e., procedures that can call other procedures). Active Values are tied heavily to graphic displays that will use demons to trigger processes when they are changed. Teitelman and Masinter (1983) advocated the use of graphics as a tool for program development:

> Finally, we have not really begun to explore the use of graphics -textures, line drawings, scanned images, even color - as a tool for program development, for example, the system might present storage in a continually adjusting bargraph, or display a complicated data structure as a network of nodes and directed arcs, perhaps even allowing the user to edit this representation directly. This is a rich area for future development.
>
> (Teitelman & Masinter, 1983, p 27)

The use of access-oriented knowledge bases for parallel AI architecture is currently under development by DARPA and Xerox Palo Alto Research Center (PARC). New developments in Active Values have used the concept as a programing technique which is independent of graphics.

(4) Logic as a Methodology for Knowledge Representation. Representation of a knowledge base by logic assumes that a programer can describe human knowledge entirely in terms of logic. The common logic representations are propositional logic and predicate calculus similar to the basis of PROLOG. Predicate calculus, the more common format, uses predicates such as "IS" to address one or more objects. Logical assertions "IS-GREEN (TANK)" (i.e., the tank is green) are true or false. Logical assertions addressed by predicates may be linked by logical connectives such as "AND," "OR," "NOT," "IMPLIES," and "EQUIVALENT." The causal form in which knowledge is represented by logical statements may be used to create a declarative semantic which is not tied to the statement's intended use. Tables, graphs, and charts may be linked with a declarative semantic type of language. The flexibility of a declarative semantic type of language allows coding of partial or incomplete information about a domain by using logical operators and variables. The programer uses variables to refer to objects in establishing a logic knowledge base. Nilsson (1980) suggested that model logic represented by worlds semantic (e.g., blocks world) could be entirely embedded in first order logic. Logic knowledge bases are processed by PROLOG-like theorem proving utilities in order to reason about the knowledge. Genesereth and Ginsberg (1985) suggested that automatic reasoning of logic knowledge bases is nondeterministic because it uses multiple inference paths before proving a desired result. A more advanced hardware system is probably needed to fully use the scope and resolution of logic knowledge bases for more general applications. Currently available hardware is found in the Japanese fifth generation PROLOG machines, but it is not for distribution outside of Japan.

c. **Search and Control in Knowledge Bases.** Search and control of the knowledge base can be considered an art form and a separate topic within the study of AI. At present, there is no generally efficient and effective technique for searching and controlling problem solving in very large knowledge bases. The representation of knowledge with a high degree of fidelity and validity is a necessary but not sufficient condition for reasoning about that knowledge. Search of a knowledge base involves determining an initial starting point (i.e., an initial state), choosing a correct path or paths (i.e., problem space) and using minimal resources and effort to derive a solution (i.e., goal state or inference).

d. **Uncertainty Management in Knowledge Representation.** Facts and rules comprising the data structures and interpretive procedures in a knowledge base are not always consistent or entirely certain. Representation of uncertainty in knowledge bases uses a certainty factor which is added to the fact or rule. The explicit use of uncertain, missing, and conflicting information in knowledge representation uses concepts from the larger domain of approximate reasoning in philosophy. Approximate reasoning is the critical notion needed to build constrained, modular software that parses (i.e., self-modifies) its own reresentation. Bobrow (1986) and Kuipers (1986) have suggested that approximate reasoning can allow a functional model of a system to explain causality in terms of the system's behavior (i.e., increments of state transitions in components or objects in relation to changes in the total system). Weld (1986) has extended this notion to the representation of aggregation in simulation. The following theories for using uncertain information are usually employed in Knowledge Representation in AI Knowledge Bases:

(1) **Fuzzy Logic:** Zadeh (1981, 1985) extended two-valued syllogistic reasoning to allow an indication of doubt in the major premise, the minor premise, or the conclusion. This was accomplished by appending arbitrary predicates which serve as quantifiers in each term of the syllogism:

In a different approach which is outlined in this paper syllogistic reasoning is employed to provide a basis for the formulation of rules for combination of evidence. Classically, the major and minor premises in syllogistic reasoning are allowed to contain only the standard quantifiers 'all' and 'some'. For example the paradigmatic syllogism Barbara is expressed by the rule

$$\begin{array}{l} \text{All A's are B's} \\ \text{All B's are C's} \\ \hline \text{All A's are C's} \end{array}$$

where A, B and C are arbitrary predicates in two-valued logic. This syllogism expresses the transitivity of set containment and, as such, provides the basis for property inheritance in knowledge representation systems.

In a more general sense which is employed in this paper, a syllogism is an inference rule of the form

$$\begin{array}{l} \text{Q1 A's are B's} \\ \text{Q2 C's are D's} \\ \hline \text{Q3 E's are F's} \end{array}$$

where Q1, Q2, and Q3 are numerical or, more generally, fuzzy quantifiers (e.g., 80 percent, most, many, few), and A, B, C, D, E, and F are crisp or fuzzy predicates (e.g., greater than 5, small, tall, heavy). The predicates A, B, ..., F are assumed to relate in a specified way, giving rise to different types of syllogisms...

(Zadeh, 1985, p 417)

Zadeh (1986) has suggested that classical probability is subsumed by fuzzy logic. This extension allows crisp and fuzzy propositions to represent the meaning of uncertain events and dependencies and allow systematic inferring from all such representations.

(2) Dempster-Shafer Theory of Evidence: This theory of evidence allows the use of confidence factors derived from a belief function based on the subjective judgment of evidence about missing or incomplete information. Shafer (1976, 1986) developed a representation methodology where belief in a proposition X was defined by the subinterval (s(X), p(X)) of the whole interval. The lower term, s(X), represents the support or minimum value for the likelihood of proposition X. The upper value, p(X), represents the most optimistic plausibility or maximum likelihood of proposition X. Dempster (1968) developed a formalism to combine reasoning from disparate sources based on an extension of Bayesian inference procedures. The formalism is called a FRAME OF DISCERNMENT (Theta) and is defined by Shafer (1986) as a list of possible answers to a question about which probability judgments are made to yield a correct answer. The Dempster-Shafer theory of evidence allows the modeling of a very ambiguous knowledge source, such as the operation of a collection of military sensors.

2-7. AI DATABASE MANAGEMENT

a. Introduction. The use of large computer simulation models requires an effort in organization, structure, and maintenance of databases. CAA uses external and internal data in several ways. In large simulations, data are preprocessed and aggregated. The results of this preprocessing and aggregation are entered as input into a large simulation model. Results from the large simulations are in turn used as input into post-processors. The postprocessors analyze and disaggregate data. The evaluation of AI methodologies for database management takes a molar (i.e., an unanalyzed mass) approach to management of large data files and Army records used at CAA.

b. Database Management Tasks. The study of computer management of large files of data is a subarea within computer science. File size and complexity have grown as an inverse function of the access speed of general purpose computers. The complexity of programing computers to quickly access and store files has outstripped the capacity of the nonexpert user to understand the process. This trend has resulted in the development of languages which buffer the user from the details of writing, reading and reorganizing, and backup of data files. The role of user-friendly query and file-management languages that give the end-user a relational view of the data files has become critical to the use and management of large data base file systems by non-experts. The needs of data base management tools according to the CAA database manager were:

(1) Intelligent access to what the model needs.

(2) Transparency (i.e. user-friendly processing of data).

(3) Efficient file manipulation.

(4) Quality assurance/error analysis.

c. AI and Database Management. The self-organizing and self-documenting features of the LIST Processing (LISP) language may also be used to control data files. The ability of LISP languages to run programs as active data structures may allow effective data base management without a high cost in human or machine efficiency. AI computer languages have allowed the development of architectures that are ideal for database management. AI languages and methodologies allow file independence, sharing, and integrity (i.e., uniformity between data and the real objects they represent). Roussopoulos (1986b) has suggested that the central issue in using knowledge bases for AI applications in database management is transfer to technology:

> I dare to predict that the dawn will be definitely gone when we all recognize that the knowledge base is the most fundamental component of any system and that all other components are engineered in and around the management and control of information.... The management of complex and continously evolving factual knowledge is very different from that of the 'snapshot' database captured in commercial

systems. The question is: how are we going to evolve from where were are to the database systems of the information age.... Ten years ago, when some of us were advocating AI techniques for better information processing some considered the idea as 'illusionary' and others as 'prophetic.' Lets hope that our visions will materialize again.

(Roussopoulos, 1986b, pp. 7-8)

The CONS structure of LISP allows access to the entire database. The address sequence and pointer sequence of the CONS allow tracking of the entire storage memory of the data base. Barr and Feigenbaum (1982) have suggested that large database facilities have always been possible in LISP and its dialects because of the property list feature of the language:

Database facilities were present in a crude form in the LISP property-list feature. The next level of complexity is represented by the multiple-index scheme of SAIL associations. Next about the use of fixed-form associations is the structural pattern-match as a general retrieval mechanism, found in PLANNER, CONNIVER, QLISP, and FUZZY.... To sum up very briefly, LISP introduced a simple and flexible data type and a way to represent functions as data. PLANNER introduced the general associative database, and CONNIVER improved it by the addition of contexts. QLISP and FUZZY increased the power of the database by defining special data types and putting everything into a discrimination net. SAIL went in another direction, developing an efficient multiple-index scheme. Finally, POP-2 showed how a wide range of data types can all be treated in a clear and uniform way.

(Barr & Feigenbaum, 1982, p 44)

Semantic node methodology (i.e., the structuring of databases as a collection of classes of objects) would allow browsing of the database and selective retrieval and processing of data within files. The use of message passing in frame-based methodologies driven by a procedural orientation (i.e., pattern-matching and demons) would allow relational, hierarchical, and network databases to be used in conjunction with an object oriented AI database. These AI methodologies would model a database around user requirements rather than machine requirements. Roussopoulos (1986a) has suggested the use of object datatypes in database technology for an Engineering Information System (EIS) application:

An EIS deals with abstract data types representing objects, Object Data Types (ODT). Many of these require specialized software for storage, display, and processing. ODTs hide the internal structure of compound objects from external access operators and leave all the intraprocessing to

> special purpose operators. Objects and inter-object relationships are stored, accessed, and managed by a database system that is suitable for handling meta-data. Schematics, graphs, images, rules, windows, texts, spreadsheets, voice, video, etc., are some of these UDTs and simulators, plotters, tailored access methods, etc., make up some of the software for processing. This list is by no means exhaustive and it only samples generic types of software which abounds current engineering applications. Redundant multiple representations of the same data is necessary to support the multiplicity of tools and users. Managing collections of these advanced objects and their meta-data is different than managing atomic data entities in a database system. Often meta-data is needed to be described by meta-meta-data and so on. Typically, meta-data, referred to as the intension of the database, is used to interpret and control the data, the extension, of the level below. The database is thought of as having multiple specification levels each of which serves as the intension of the next level's extension. Convenient languages with recursion for accessing and manipulating objects at all levels of this hierarchical specification are needed.
> (Roussopoulos, 1986a, pp 1-2)

d. Future Directions. The management of databases by the user rather than by machine requirements offers many advantages to CAA. The incorporation of data bases within knowledge bases would allow multiple user requirements to be incorporated into database management. AI graphics, menus and operators to control the software and the machine would allow assistance to the end user in query of the database. This would allow CAA study users to participate with application programers in database management. In turn this would allow a dynamic model of data which would reflect the real world of data (i.e., the end user would incorporate his logic and requirements into the management of the database). Project AMBER developed by Intellicorp and released in January 1987 is an example of LISP based software which allows the mapping (i.e., transforming information from one form to another) of a database into a knowledgebase using a frame representation. This database management tool allows interactive browsing of the knowledge and query by reformulation of the knowledgebase.

e. Database Issues for Military Modeling and Simulation

(1) Integration of Knowledge Base Techniques with Existing Data. Both corporate and military communities are wedded to numeric data bases. The transition to models that address the same databases in conjunction with knowledge-based systems will be difficult. If the amount of investment in traditional databases is considered, a more practical approach will be to use knowledge-based techniques to address existing data bases.

(2) Multiple Knowledge Sources. Military knowledge tends to be organized into doctrine. The representation of knowledge from multiple domains remains a difficult problem for present knowledge representation techniques. The problem of weighting multiple sources of knowledge according to their validity or importance is a current research issue within the AI community.

(3) Missing Information. The absence of data or information often provides a clue to human problem solvers. There does not appear to be a methodology capable of representing missing knowledge within a total knowledge representation scheme at the present time.

(4) Nonspecific Search (browsing). The ability to browse database information which will later become organized is important to managerial level end-users. Larson (1986) suggested that browsing frees a user from the prerequisites of database management systems. Browsing can replace or supplement the requirements that the user understand the names and types of objects being searched as well as enabling notions of database logic structure. Intelligent database software is not available which will allow the users' ill-defined notions of what they desire and the vague understanding of the data to be used in structuring the search of the database. The search of nondomain-specific information in very large databases or knowledge bases used in conjunction with databases will be a critical problem for large-scale applications of AI techniques. Jakobson, et al. (1986) created a set of software packages that serves as an intelligent database assistant. The software serves as a front-end for multiple databases. Database expertise is written into the software and mixed with sophisticated user utilities such as natural language queries (i.e., typed English sentence input into software windows with mouse controls). Milne (1985a&b) suggested that Army Information Management expert systems require an emphasis on displaying the information in order that a commander may view the implications of a possible choice without the system making the decision. It is clear that very sophisticated browsing capabilities will be required in any future systems for military simulation. Systems of stored facts and data will be encapsulated by answer-generating code that allows an end-user to gain maximum insight from a simulation.

2-8. EXPERT SYSTEMS IN AI SIMULATION

a. Overview

(1) Introduction. Expert systems are an applied subfield of AI programing. In essence, expert system software mimics the reasoning processes of human experts in limited topic areas. The software is designed to perform problem solving in a specialized (i.e., bounded) area of expertise called a domain. The domain knowledge is encoded in the knowledge base of the software system. Expert system software is somewhat like word processing software. Parts of the software read a database of raw text (i.e., a knowledge base) and provide structure (through an inference engine) to yield a formatted product (i.e., an inference). Expert systems can be conceived of as a programing methodology which produces computer software that mimics the reasoning processes of human experts, sometimes surpassing it, for limited topic areas. Expert system software has been used with great success in problem solving within narrow domains of human knowledge.

(2) **Analysis of Expert Systems.** Expert system software consists of (1) a knowledge base; (2) an inference engine; and (3) users' interfaces. Waterman (1986) has listed five steps in building expert system software: (1) problem identification; (2) system conceptualization; (3) formalization; (4) implementation; and (5) testing. The CAA purpose for using expert system software would be to capture expertise in simulation models and the study process.

(3) **Expertise in Expert Systems.** In a review of the acquisition of expertise, Lesgold (1983) has suggested that the process is dynamic and not fixed. This notion may have serious implications for military modeling applications. The use of expert system software may be the only known method of codifying some military knowledge (i.e., doctrine) in a usable format for simulation. Expert systems appear to be required for flexible problem solving in large simulation models. They can deal with some kinds of uncertainity and complexity better than any alternative now in existence. A distinct possiblity exists in the near future to use expert system software to fuse real-time sensor and intelligence data with stored knowledge bases to yield an online operation decision-aiding system for military commanders at all levels of command. Hayes-Roth (1981b) has suggested six functions that expert system software can now perform:

- Use expert rules to avoid blind search.
- Reason by manipulating symbols.
- Grasp fundamental domain principles and weaker general methods.
- Solve complex problems well.
- Interact intelligibly with users.
- Interpret, diagnose, predict, instruct, monitor, analyze, consult, plan, or design.

(Hayes-Roth, 1981b, p 106)

b. Components of an Expert System

(1) **The Basic System.** The basic components of an expert software system are a knowledge base and an inference engine. The basic notion is that a knowledge base is searched by an inference engine for the answer to a problem. Figure 2-3 illustrates the basic software that is required to mimic a "cognitive process" in machine intelligence. Knowledge (i.e., judgments, facts, rules of thumb) from the human is stored in the knowledge base and is extracted and organized by the inference engine.

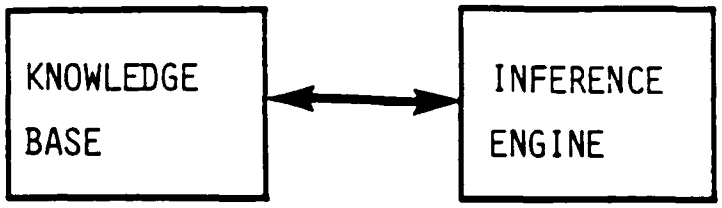


Figure 2-3. The Basic Expert System

The knowledge base is normally tailored to a narrow domain of expertise. The inference engine serves as an interpreter of the knowledge base program. Together the system can be used to make inferences about new data.

(2) **Complex Expert System Software.** Complex software components in expert systems are generally difficult to operate by the end-user without machine and software support. Most expert systems in industrial use have many supporting utilities. Meyers (1986) expanded the basic notion of the expert system software to include utilities that make the system easier to use by nonprogramers. This system is illustrated in Figure 2-4. Two additional subcomponents are added to each part of the basic expert system. The knowledge base was expanded to include (1) assertions about the problem (i.e., declarative knowledge in working memory), and (2) knowledge relationships (i.e., formulas about the interrelationships of the

information within the knowledge base). The inference engine was expanded to include (1) search strategies (i.e., how to look at the knowledge base for information needed to make inferences), and (2) an explanation tracing utility (i.e., software to provide an audit trail of how an inference was made). This system is enhanced by heavy interactive graphic routines and a natural language preprocessor to allow nonprogramers to query or provide information to the system in plain English text. The tracing utility and the graphics are useful in development and refinement of expert systems. The use of complex expert systems has the potential for allowing military decisionmakers to directly control simulation methodologies through powerful man/machine interfaces. Zadeh (1985) has suggested that fuzzy qualifiers (i.e., multivalued or fuzzy logic) can provide a methodology for the management of uncertainty in expert systems. Conventional two-valued syllogistic logic (i.e., categorical syllogisms) have all, some, or no quantifiers replaced with multivalued or fuzzy quantifiers such as many, few, 1-99 percent, etc.

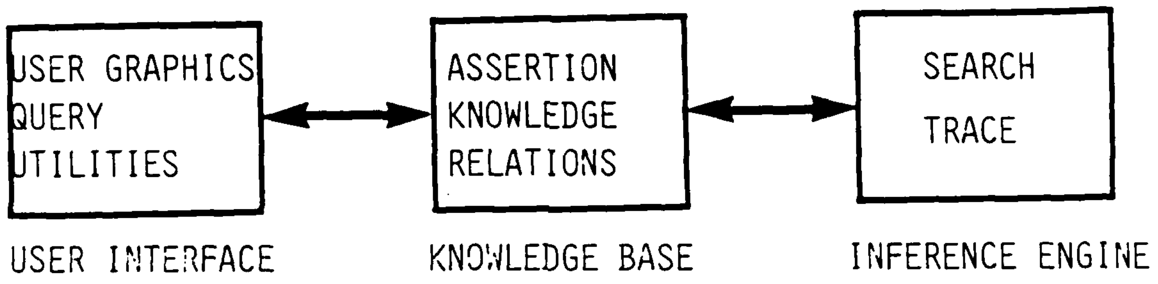


Figure 2-4. Complex Expert System Software

c. **Development of Expert Systems.** The development of expert system software generally follows the principles of good software development. Expert system software differs from general software development in its output. Expert system software output is congruent but not isomorphic to real-world problem solving. Figure 2-5 illustrates the interrelationship of real-world problem solving and expert system software output.

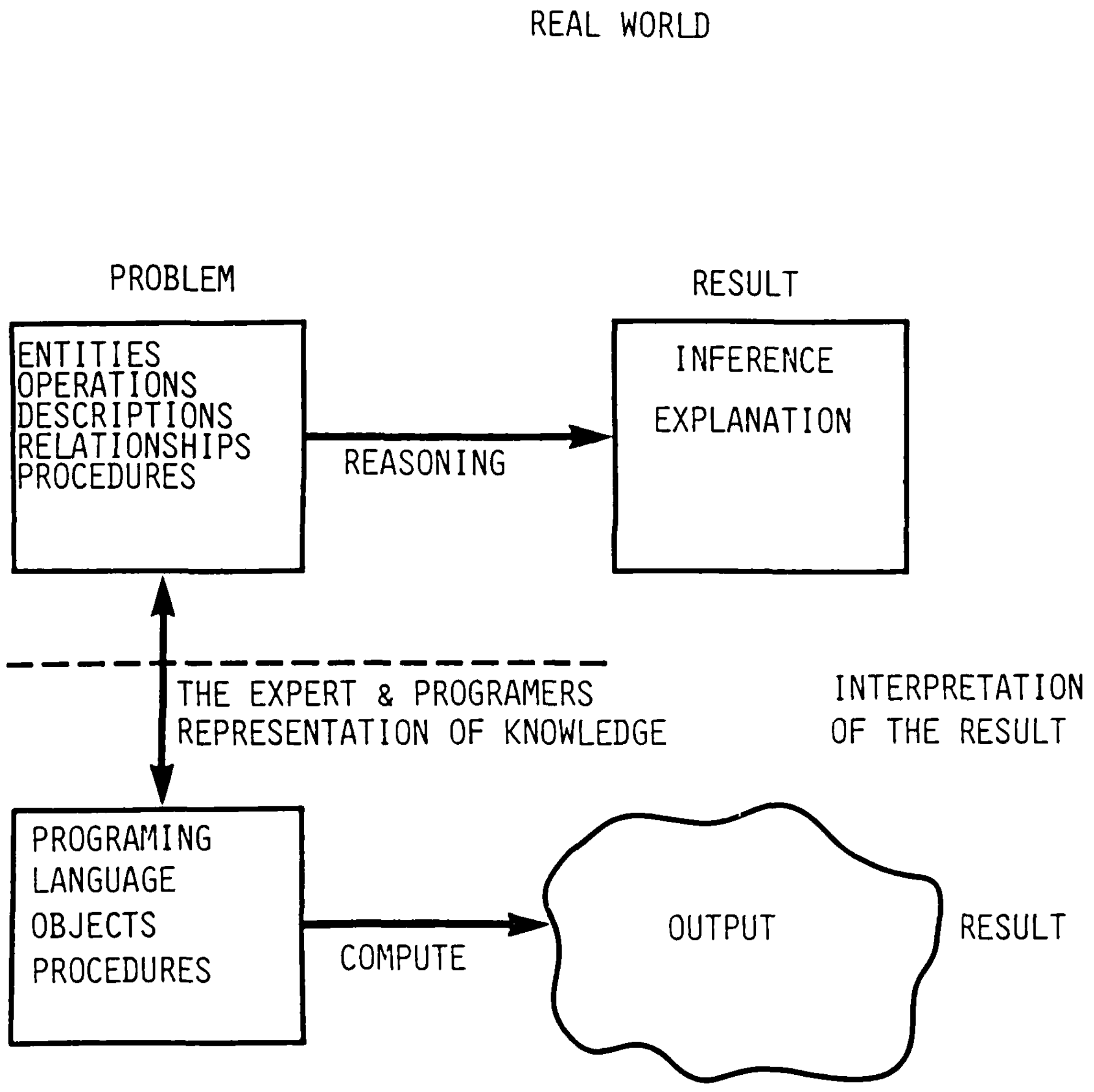


Figure 2-5. The Real World vs Machine Problem Solving

Denning (1986) suggested the following distinction be made between shallow and deep expert systems:

> Shallow systems are designed for speed. In their limited databases they store more facts than rules. The proofs of their conclusions are usually short, and most of the conclusions strike observers as being straightforward consequence of information in their databases. They often give poor results when applied to problems other than those for which tney have been tested. Examples include a system to generate production schedules using bin-packing heuristics and a system to answer electronic inquiries for technical information. Deep systems derive their conclusions from models of phenomena in their domains, using first principles embedded in the model. The proofs of their conclusions are usually long, and some conclusions may strike some observers as anything but obvious. An example is Dendral, a system that when given spectral data, generates diagrams of chemical compounds using ball-and-stick atomic models. Other prominent examples include qualitative physics programs which determine states of behavior of physical systems.
>
> (Denning, 1986, p 81)

Both the programer's representation of the problem and the end-user's interpretation of the result must develop over time. Hayes-Roth, Waterman, and Lenat (1983) have suggested an incremental technique to build expert system software. The technique consists of: (1) identification (scoping of problem requirements); (2) conceptualization (finding concepts to represent knowledge); (3) formalization (designing structure to organize the software development effort); (4) implementation (formulating software); and (5) testing (validating software). The capture of human expertise in software may require severe delimitation (i.e., bounding) of the problem and will affect the generalizability of the result. Since expert system software captures a snapshot of (hopefully) fully developed expertise, great care must be taken to build all the areas of knowledge required to solve problems in the target domain.

d. Current Expert System Applications for Military Simulation

(1) A reasonably long-term goal of expert system software is real-time inference on the battlefield. Simulated systems are a necessary step toward this goal. Hayes-Roth (1981b) suggested AI applications to modeling and simulation involved the addition of new capabilities, languages, and architectures to the discipline of military operations research. The new capabilities focus on methodologies which are nonprocedural means to manage entities and involve the end-user in the model building process. Languages such as LISP approach encoding in English-like words and provide an understandable explanation of the system's action. The most revolutionary addition to modeling and simulation is in the new architectures available. Improvements in the architecture of simulation allow models to focus simulations toward specific questions and compute selected data to provide output which is relevant to the question. At the present time, expert systems can quickly apply simple knowledge to measurements of combat and rapidly reach an inference about the underlying combat conditions. This gives the simulator or model user aggregates of higher level information. The ability of the software to encode self-knowledge of its deep structure and operation creates an audit trail for the user to gain a detailed understanding of the decision rules of the simulation. This characteristic of expert system software to encode self-knowledge could lead to self-modification based on the results of the simulation (i.e., learning). At present, expert system software will allow the model logic of military simulation to approach human thought and judgment more closely than any other technique.

(2) Hayes-Roth, Waterman, and Lenat (1983) have developed a classification scheme of 10 categories of specialized problem solving expertise currently programable on expert systems. These systems can:

(a) Interpret and infer situations from sensor input.

(b) Predict and infer consequences (i.e., what-if).

(c) Diagnose and infer errors and breakdowns from sensed and observed data and knowledge.

(d) Design a configuration under constraints.

(e) Plan and design actions.

(f) Monitor by comparing measured and observed data to vulnerabilities in plans.

(g) Debug and prescribe fixes for breakdowns.

(h) Repair using plans that prescribe a fix or solution.

(i) Instruct by diagnosing, debugging, and repairing of learned behavior.

(j) Control systems by interpreting, predicting, repairing, and monitoring system behaviors.

(3) Kline and Dolins (1985) drafted 47 guidelines for chosing AI techniques in developing expert system software. The taxonomy found in Appendix F may be a useful guideline for developing expert system software in a military and general purpose environments.

(4) A well-managed knowledge base and expert system library would allow a model builder to structure pieces of existing expert system software into unique software systems for problem solving simulations at CAA. DARPA is presently conducting experiments with networks of expert systems.

2-9. AI TOOLKITS

a. Introduction. Toolkits are computer programs that are used to build AI software. The most common application of a toolkit is building expert system software. Toolkit software typically contains an inference engine, knowledge acquisition aids, and user/programer helper utilities. AI toolkits create a development environment. Ideally the toolkit can also link a programing environment (hardware, software, and programer) with the user as shown in Figure 2-6.

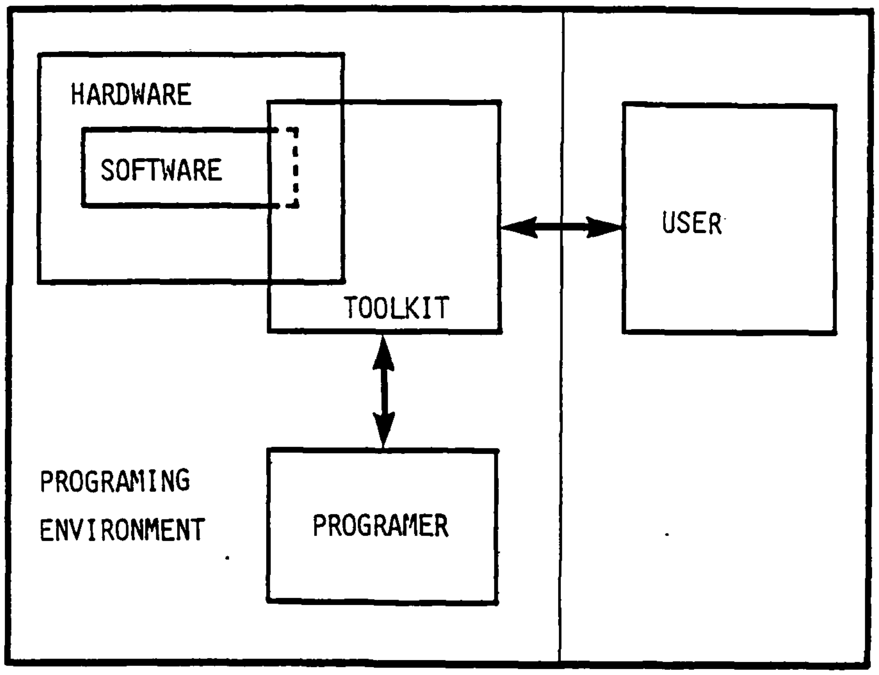


Figure 2-6 Artificial Intelligence Toolkit Environment

If large complex LISP programs are built with many programers using small and sometimes interrelated functions, a toolkit may be required. Toolkits are also useful for testing and documenting LISP programs. The effect of a toolkit as part of a total development/production environment was clearly defined by Teitelman (1969):

> In normal usage the word 'environment' refers to the aggregate of social and cultural conditions that influence the life of an individual. The programer's environment influences, and to a large extent determines what sort of problems can (and will want to) be tackled, how far he can go, and how fast. If the environment is cooperative and helpful (the anthropomorphism is deliberate), the programer can be more ambitious and p oductive. If not, he will spend

> most of his time and energy fighting a system that at times seems bent on frustrating his best efforts.
> (Teitelman, 1969, p 5)

Design consideration of an AI development or production environment needs to work backward from the study sponsor(s). The action officer, study director, programer, toolkit, software, and hardware all need to be considered in tailoring an environment for the development and production of AI software. In effect, this would provide a custom design for each study effort. A suitable starting point to develop a total environment for AI would be in the area of human factors engineering. The Human Engineering Guide to Equipment Design (VanCott & Kinkade, 1972), the Army's Human Engineering Guidelines for Management Information Systems (Hendricks, Kolduff, Brooks, Marshak, & Doyle, 1983), and the Psychology of Human Computer Interaction (Card, Morgan, & Newell, 1983) provide suitable starting points for this type of design effort.

The AI toolkit provides a powerful starting point for custom design of simulations and studies. The AI toolkit allows the scoping of a study effort and the design and development of software that is unique to the end-user's purpose and cognitive problem-solving style. The organization of the end-user's cognitive style of problem solving, problem conceptualization, problem-solving strategies, and solution utilization can be incorporated into the study effort with AI toolkit software. McCune and Dean (1985) suggested that the toolkits will allow reasoning about the domain of application in order to tailor the product to the user:

> Future tools will have the ability to be highly tailored to suit the needs of the particular situation, including management hierarchy, application domain, other tools available in the programing environment, and idiosyncracies of tool users. Obviously, this level of variability goes well beyond the simple parameterization or runtime options found in many tools today. Modeling large bodies of facts and preferences requires knowledge representation from AI. Modifying these bodies requires an ability to elicit new or modified knowledge from users or to learn by observation. These areas are among the most difficult in AI research, but the potential is tremendous.
> (McCune & Dean, 1985, p 4)

Toolkits provide a powerful environment for rapid prototyping of more complex software. The user interfaces provide the capability for the sponsor to input his thought pattern, problem-solving style, and personality directly into the program structure. Harmon and King (1985) developed three categories for evaluating AI toolkits:

(1) **Small System Building Tools.** Tools that can be run on personal computers. These tools are generally designed to facilitate the development of systems containing less than 400 rules.

(2) **Large Narrow System Building Tools.** Tools that run on LISP machines or larger computers and are designed to build systems that contain 500 to several thousand rules but are constrained to one general consultation paradigm.

(3) **Large Hybrid System Building Tools.** Tools that run on LISP machines or larger computers and are designed to build systems that contain 500 to several thousand rules and can include the features of several different consultation paradigms.

(Harmon & King, 1985 p 92)

The assessment of toolkits was delimited to large hybrid system building tools and large narrow system building tools. Small system building tools were assessed to be inappropriate for CAA's large models although some supporting tasks that aid model development and simulation could be supported by microcomputer AI toolkits. The use of AI toolkits for report documentation and publication may provide valuable assistance to study directors and management support in the study processes. The use of large hybrid system building tools for CAA prototypes is desirable for several reasons. First, it allows users to become familiar with the maximum capacity of both the software and the hardware that is required to grow toy problems to industrial-sized (i.e., CAA-sized) production-level expert systems. Second, there is a savings in "learning cost" since personnel have to acquire proficiency on only one system.

b. **Large Narrow System Building Tools.** This class of toolkits has been adapted from expert systems that have been previously built. The knowledge base (i.e., rules) have been extracted and modified, and the expert system becomes a toolkit to build other expert systems. The remaining interpreter, scheduler, and inference engine have usually been optimized on a specific knowledge base. This narrow development of the toolkit may restrict its applicability (i.e., generalizability). The following large narrow systems were judged to be among the most applicable to CAA and its mission:

(1) **MACSYMA.** MACSYMA was developed as a computer algebra software system at the Massachusetts Institute of Technology (MIT). MACSYMA is written in LISP and can be run in an interactive or batch mode. MACSYMA allows a user to compute complex mathematical problems such as indefinite and definite integrals, and discrete Fourier transforms. The user may solve algebraic equations, systems of linear equations, and systems of nonlinear equations. Parts of the software also address solution of differential equations. The toolkit allows the users to focus on the abstract nature of their problem and address its procedural aspects while allowing the software to solve the routine computational aspects of the problem. The requirement that the users explicitly specify the representation of an expression as input allows users to gain relational insights into the problems they are solving. The results of the problem can bechecked by the software. There is also the option to generate FORTRAN code automatically. Pavelle, Rothstein, and

Fitch (1981) evaluated MACSYMA and concluded that a wide variety of mathematical applications are possible with this toolkit. Symbolics (1985a) has cited over 3,000 applications in government and industry that have used the MACSYMA toolkit.

(2) **TIMM.** The Intelligent Machine Model (TIMM) is a FORTRAN-based toolkit for programing knowledge acquisition. TIMM is an induction-based system that is menu driven and is very user friendly. The user enters a large number of examples into the program. The software converts the examples into a rule and proceeds to build a simple rule-based system. The assumption of the toolkit authors was that many simple rule-based systems could be linked together to form a powerful expert system. Cooper, Kiselewich, and Weeks (1985) argued that such a system is sufficient for knowledge representation and extraction of pattern from global data bases (i.e., pattern matching). Since patterns are nested, FORTRAN appears to be a difficult implementation language for knowledge representation and programing search strategies for large expert systems. TIMM appears to be a useful tool in the initial stages of building a large expert system, but a user would need another tool to complete the job. CAA used TIMM to build a prototype automated decision module for the McClintic Theater Model (Cooper and Pennington, 1983a). The module consisted of 10 submodules containing 263 FORTRAN subroutines that were built around TIMM and the wargame. Cooper and Pennington (1983b) suggested that a great number of low-level support routines were needed to provide utility services that interacted with the wargame. The knowledge base of the prototype contained a knowledge body (Cooper, & Pennington, 1983c) which consisted of stored sets of rules listed in tables. These rules were used for decisions in the wargame and post-decision analysis.

(3) **RULEMASTER.** RULEMASTER software assists the programer in structuring the knowledge base of an expert system program. The toolkit has a component for inducing rules from examples. A separate component is also available for expressing the rules in an organized format. The knowledge base is developed using both declarative knowledge (i.e., simulated situation states and theorem provers) and procedural knowledge. Procedural knowledge is used to structure condition-action pairs of rules to allow the program to reach a goal or solve a problem. Facts and rules that are uncertain are incorporated with fuzzy logic (i.e., probabilistic monotonic logic). RULEMASTER has been used in developing an expert system program to aid in forecasting violent thunderstorms (Riese & Zubrick, 1985) and to aid decisionmaking in environmental regulation (Hadden & Hadden, 1985). RULEMASTER is programed in C language. The software will run on either the VAX or the SUN hardware systems.

(4) **OPS 5.** The Official Production System version 5 (OPS 5) was developed at Carnegie-Mellon University. Forgy (1981) developed OPS 5 under DARPA contract to provide a tool for programing both knowledge and control structures for cognitive modeling (i.e., human memory and problem solving). The toolkit uses a production system architecture. This system employs a set of rules called production memory in a global database (i.e., data memory). An inference engine executes the production cycles by

pattern matching. Brownston, Farrell, Kant, and Martin (1985) suggested that OPS 5 serve as the basis for a family of production system programing tools that can be used for research and development. OPS 5 uses BLISS, MACLISP, or FRANZ LISP for its interpreters. The OPS 5 toolkit is presently on CAA's VAX 11-780 hardware.

(5) **PICON.** Process Intelligent Control (PICON) was developed by LISP Machine, Inc. (LMI). It has been used as a software development toolkit for building expert systems that monitor industrial process control in real-time simulations. The toolkit is frame-based in that it can represent a single concept by properties that ascribe it (i.e., the defining attributes of the concept). Additionally, the software system can use rules to structure the knowledge base. Expert system software built with PICON is used to monitor process interruption of large complex systems such as a chemical plant or nuclear reactor. Systems that monitor up to 20,000 variables can be built with PICON. A knowledge base of over 500 rules can be structured with the PICON software toolkit. There is a graphics package available that can assist in structuring the knowledge base. PICON also has a simulation capability. PICON is currently written in ZETALISP and C languages and will run on Symbolics or LMI hardware.

(6) **NOTECARDS.** NOTECARDS was developed by the Xerox Palo Alto Research Center (PARC). The toolkit was written with very friendly user interfaces to assist a nonprogramer in structuring knowledge. The program allows the user to mix text, digitized maps, graphs, pictures, and charts with text and graphics. The software will allow the user to tie nodes of material together with user-defined interrelationships. The inference engine will search the knowledge base and yield causal inferences. The toolkit is programed in INTERLISP-D and runs on the Xerox D-series (Dandelion, Dolphin, and Dorado) LISP machines. DARPA has developed a Golden Tiger Workstation (XEROX 1186 AI machine with a Notecards toolkit and SMALLTALK-80 style LISP). This workstation has been used by the CIA for intelligence analysis and studies. The laser-graphics printer, when added to the workstation, allows the output of publication quality text, graphics and illustrations. The Golden Tiger Workstation has provided an intelligent study director workstation for both study and analysis purposes.

(7) **PERSONAL CONSULTANT.** The PERSONAL CONSULTANT software toolkit was developed by Texas Instruments (TI) to aid a programer in rapid prototyping of expert systems. The toolkit was based on the classic medical expert system called EMYCIN. The system production rules are tied to EMYCIN logic. The programer that uses the PERSONAL CONSULTANT does not need to know LISP if he can narrow the problem to fit the predefined functions already embedded in the software. Under these circumstances, the system is sufficient to structure the knowledge base. There is also a frame-based system available for building a knowledge base. The software has many utilities to help the programer build an expert system. Color, traceback subroutines, and a regression analysis package would probably be useful in developing simulation applications. The inference engine can develop an explanation of its forward and backward rule chains. A class

inheritance methodology is used to structure the rules. The PERSONAL CONSULTANT has aided programers in building expert system software that contains up to 400 rules. Most of the current applications of the toolkit have been in building expert systems in the business analysis area. Expert systems for policy and rate selection for insurance (i.e., risk analysis) and equipment fault diagnosis have been built with PERSONAL CONSULTANT. The software toolkit is written in LISP and runs on a TI Explorer LISP machine. A microcomputer version has also been written for the TI business microcomputer and the IBM PC.

c. **Large Hybrid System Building Tools.** This class of software tools allows the programer to develop applications at the leading edge of technology. Rule-based systems of over 500 rules can be built and documented using this class of software. The construction of a theater-level expert system with any degree of resolution will probably require the use of this software. It appears that these toolkits, although expensive, are a cost-effective means of prototyping large and complex decision support systems for combat analysis. The use of a toolkit may aid the scoping of large systems that need to be refined at a later date. The use of such a software tool is a necessary but not sufficient condition for building a valid theater-level expert system within time and budget constraints usually imposed on CAA. The following large hybrid system building software toolkits were evaluated as the most appropriate for CAA's mission:

(1) **KEE.** The Knowledge Engineering Environment (KEE) is the premier software toolkit of large hybrid systems. KEE evolved from an expert system that was used for genetic research. KEE supports all major methodologies for knowledge representation (frames; production rules; procedural orientations; and object orientations). The frame block attributes may be varied by the programer. This restriction of values on slots (i.e., attributes of a concept) when used in inheritance (i.e., values of a frame that are passed on to a more detailed resolution) provides a constraint methodology for searching the knowledge base. A worlds feature allows multiple hypothetical situations to be represented in a manner that allows merger, comparison and contrast. The inference engine can use forward or backward chaining. Active values (demons) and variables are used in the rule system. The graphics use icons (active images) to change values in the knowledge base and show these changes on the computer screen. The toolkit may be extended with a software package called SIMKIT. This extension allows the programer to create and verify a model. The SIMKIT allows expert system software to input different scenarios, and make decisions within the simulation while the simulation is in progress. The programer may also use SIMKIT to analyze the output of the simulation. Both continuous and discrete event simulations can be built with SIMKIT. Continuous probability distributions (normal, uniform, exponential, and lognormal) and discrete probability distributions (Poisson and step) can be generated within SIMKIT and tied to slots in the frame representation methodology. KEE was written in INTERLISP and has been updated in common LISP. It runs on all AI hardware systems. Mainframe and microcomputer versions of KEE are available. Friedland and Kedes (1985) used KEE to model a DNA molecule and analyze gene sequences. Although the structure of

DNA has been proximally defined, the degree of complexity and uncertainty in modeling the DNA molecule and its function at the chemical level may approach the problems in modeling land combat. Faught (1986) has reviewed the current applications of KEE to engineering problems.

(2) **ART.** The Automated Reasoning Tool (ART) allows rule-based knowledge representation with options for frame and procedural methods of knowledge base representation. ART evolved from an expert system used to interpret radar signals from space flight operation at NASA. Its inference engine allows both forward and backward chaining. It also contains a viewpoint subroutine which allows temporal and hypothetical reasoning in the knowledge base. Hypothetical reasoning allows the inclusion of contradictory solutions in the inference process. Temporal reasoning allows the modeling of time states. A planning and scheduling subroutine reduces search space in the knowledge base compiler which produces fast running procedural code and indexed databases. ART was programed in LISP and runs on LMI, TI, and Symbolics AI machines. A version is also available on the VAX, and a future release will run on the SUN. Production versions of expert systems developed on ART have been adapted to run on the IBM PC.

(3) **SRL.** The Schema Representation Language (SRL) software toolkit is now called KNOWLEDGECRAFT. The software for developing the knowledge base uses a frame-based methodology for knowledge representation called schemas. The software was originally designed for expert system development in robotics. Object-oriented, rule-based, and logic-based knowledge representation methodologies are also included. The schema method can use a metaknowledge methodology which organizes frames by utilization. Slot values may also be organized within frames by using the metaknowledge methodology. Procedural attachment of functions to slots may be programed by using the demon subroutines that are also part of the software package. The inference engine has forward and backward chaining mechanisms. A user-designed search algorithm is also allowed. The software has a discrete simulation language embedded within the software toolkit. The error subroutine can be modified by the programer to allow sensitivity of the system to vary. The graphics subroutines allow zooming on various levels of resolution using the mouse. Windows may be programed to allow two-dimensional transformations and scrolling of information within a window. Unique to toolkits of this size and scope is a natural language subroutine that provides an interpreter to allow English, French, and German text to be typed into the system and transformed into code. The software is written in COMMON LISP and will run on LMI or Symbolics LISP machines. A version for the VAX 11-780 using the VMS operating system is available. The software toolkit has been used in the development of a manufacturing forecasting system for logistic management.

(4) **LOOPS.** LOOPS is a research prototype toolkit that is sold but not presently supported by the Xerox company. The software toolkit was developed at the Palo Alto Research Center (PARC). Bobrow and Stefik (1981) suggested that the goal of LOOPS was to make the programing paradigms of knowledge engineering simple and easy to learn. The software

allows procedural-oriented, object-oriented, access-oriented, and rule-oriented knowledge-based structuring methodologies. The software also allows the programing paradigms to be used in conjunction with each other or to be integrated together for programing and debugging knowledge bases. The procedural methodology allows inheritance in the knowledge base to be displayed graphically with active values. This allows interactive browsing with a terminal screen. The program can modify or edit objects in the knowledge base and change inheritance relationships. The software toolkit also contains a rule compiler that keeps the LISP code in correspondence with the rules and allows the compiled program to run faster. The toolkit has extensive support for debugging. The software is programed in INTERLISP-D and runs on the Xerox D-series LISP machines. Stefik, Bobrow, Mittal, and Conway (1983) developed an economic application using LOOPS to simulate independent truckers for instructional purposes.

Section II. ARTIFICIAL INTELLIGENCE HARDWARE

2-10. AI ARCHITECTURE

a. Overview. The original symbolic computing was done on an IBM model 704 (Levy, 1984). Due to the requirements for large amounts of main memory and processors, a special class of hardware was developed. Bawden, et al. (1979) have suggested that the advantage of a LISP machine is to give the user the ultimate personal computer. The user is given full use of the main memory, the throughput capacity of the processor, and the disk. The computer allocates to the user this processor from a pool of processors, each tied to its own piece of main memory and disk. When the user is finished, all these elements return to the pool in the machine:

> The LISP Machine is a personal computer. Personal computing means that the processor and main memory are not time-division multiplexed; instead each person gets his own. The personal computation system consists of a pool of processors, each with its own main memory, and its own disk for swapping. When a user logs in, he is assigned a processor, and he has exclusive use of it for the duration of the session. When he logs out, the processor is returned to the pool, for the next person to use. This way, there is no competition from other users for memory; the pages the user is frequently referring to remain in core; and swapping overhead is considerably reduced. Thus, the LISP Machine solves a basic problem of timesharing LISP systems.
>
> (Bawden, et al., 1979, p 348)

b. The MIT LISP Machine

(1) LISP machines grew out of a project started at the MIT computer science department. The first AI machine was developed at the MIT AI laboratory in 1976. This machine was called the CONS. The CONS was a diskless machine that had to be collocated with a PDP-11 for I/O and memory. The CONS machine evolved into the CADR machine. The CADR machine can be considered the first standalone AI machine. The CADR runs a 32-bit microprocessor using a paged virtual memory with 256 words per page. The CADR had a recursive descent compiler. This allowed AI programers to back into or out of execution orders in their software.

(2) LISP words on a MIT LISP machine are larger than those on most general purpose computers. The LISP machine uses a 32-bit word that is CDR-coded to reduce memory storage. All versions of LISP use pairs of pointers in memory which are stored jointly. CDR coding depends on LISP list structure which has a two-part list called the CONS. The first element of the CONS is called the CAR. The rest of the list is called the CDR. Two bits of the LISP machine word are used to indicate the value of a CONS pair (i.e., the two-word list element is the CAR of a CONS, which is followed by a CDR) and their storage in the memory of the LISP machine. Tne CDR-code field may have the value of "CDR-NORMAL, "CDR-NEXT," and "CDR-NIL." Regular storage is given the value of "CDR-NORMAL." Compacted LISP code is given the value "CDR-NEXT." The last CONS is given the value "CDR-NIL." This has become the standard notion for LISP words.

(3) Much of the power of a LISP machine is gained through its architecture that is designed to handle function calls. In LISP, everything is effected with function. Function calls on a list use the CONS on the LISP machine to gain efficiency. The function call and function return features of the LISP machine set it apart from general purpose computers. The LISP machine and its software will allow a function to take arguments and to return a LISP object. Variables referenced by the function may be bound or set by the user differently. Calling a function opens a stack buffer in the machine. The programer can push arguments onto the stack in a nested fashion similar to APL. This process is normally not visible to the programer. The functions that "push" or "pop" elements onto or from the front of a list which is stored in global type variable are very useful for program development and debugging. All LISP expressions are functions. Stacked hardware allows recursive function calling. Gabriel (1985) described the advantages of stacked hardware for function calling in a LISP machine.

> During a function call, the arguments are computed and then pushed onto the temporary stack. The function call instruction specifies the function to be called, the number of arguments, and what to do with the results (ignore them, push them on the temporary stack, return them as the caller's result, or accept multiple values on the stack). The caller sees if there is room in the stack buffer for a new frame, checks for the right number of arguments, and the legality of the function being called. If all is well, the caller builds a frame and copies the new arguments to it. The frame itself may be up to 220 words long. &REST arguments are handled by passing the callee a pointer to a CDR-coded list of the arguments from earlier in the stack (created either by the current caller or by some previous caller). This avoids the necessity of variable length data

> in the main part of the frame. The caller then transfers control to the called function, which ultimately issues a return instruction. The return instruction insures that the caller's frame is in the stack buffer, removes the called function's frame, and replaces the caller's copy of the arguments with multiple returned values, if any; otherwise, it places the single returned value in the appropriate place.
>
> (Gabriel, 1985, p 39)

Therefore, LISP machines have the advantage of using stacked hardware managed by processor hardware that moves arguments to registers that call and return functions. Gabriel (1985) has also suggested that most LISP implementations use call and return of functions in 25 percent of the total execution time.

c. Architecture in Support of LISP

(1) Much of the flexibility of LISP on a LISP machine is derived from the use of pointers. Pointers are addresses in memory registers. A large amount of memory is required to run LISP programs. Since LISP pointers are addresses in physical memory registers and LISP memory contains nested pointers and data, the use of memory tags is needed to distinguish pointers from data. The tags of arguments call the right routines by microcoded instruction sets. Matthews, Manuel, and Krueger (1986) suggested that special hardware features are required to fully exploit the LISP language.

> The system designer or software developer moving into symbolic processing soon finds that choosing an intelligent workstation requires a knowledge of the unique features of LISP architecture. These features include the extensive use of multiple variables and data types, the use of garbage collection to control data traffic, a linear address space that uses a large number of data structures, and an extended data-tagging system field. A LISP machine that effectively implements these features in hardware takes advantage of the idiosyncrasies of the LISP environment. To implement the internal LISP features, the LISP machine must have a very large microcode memory. It must also have a variety of extra internal registers to take care of data tagging. Finally, the machine must have a dual-pointer stack to track data movement within the large abstract address.
>
> (Matthews, Manuel, & Kruger, 1986, p 95)

(2) Memory is a computationally expensive item (in both machine cycles and physical space) in processing LISP software in conventional computers. This is less costly in LISP machines because of the use of pointers. Matthews, Manual, and Krueger (1986) pointed out that special hardware features designed into LISP machines allow memory requirements of LISP not to limit the utility of the language:

> A LISP machine has a very large microcode memory (typically 16 Kbytes x 50 to 60 bits). In such a machine, LISP source code is not executed directly, but is compiled down to a virtual machine code (macrocode). The microcode then interprets the macrocode. To allow interpreted execution of LISP source code, a compiled LISP interpreter program is always resident in the system. Many of the internal LISP features, such as the virtual memory management and garbage collection are implemented in microcode. Since the microcode is resident in the system changes, executions and modifications can be easily made. In the data paths of a simple LISP machine, a register file drives one input of the ALU, and the other ALU input is driven by a bus. On this bus is a second register file, stack cache, virtual memory address register, memory data register, LISP macrocode location counter and macrocode instruction buffer. The ALU output drives the machine's main data bus. This bus feeds data back to the data path sources and to parts of the control section. In parallel with the ALU is a shifter/masker which also drives the main data bus. Each machine instruction uses the ALU or the shifter/masker to perform its operation. To eliminate a separate tag processor, tags are implemented as the top few bits of the data word and the normal data paths are used for tag processing. The shifter/masker is added to make tag processing, which involves many bit manipulations, more efficient. The shifter/masker consists of a barrel shifter and a full width programable AND gate. The barrel shifter shifts a data word any number of bits in one operation, and the masker allows any number of bits in the shifted word to be masked off. It then combines the masked word with a back-abstract address space in a real-word machine that has limited physical memory space, a variety of techniques must be used to implement memory management. These techniques are collectively known as garbage collection.
> (Matthews, Manuel, & Krueger, 1986 pp 97-98)

LISP pointers need to include the memory location and the memory partition. Each LISP pointer contains a virtual memory address which is converted into physical memory. Gabriel (1985) suggested that the tagged architecture and encoding of pointers allows runtime management of typing by the LISP machine:

> LISP supports a runtime typing system. This means that at runtime it is possible to determine the type of an object and take various actions depending on that type. The typing information accounts for a significant amount of the complexity of an implementation; type decoding can be a frequent operation . . .. There is a spectrum of methods for encoding the type of a LISP object and the following are

two extremes: the typing information can be encoded in the pointer or it can be encoded in the object . . . . In tagged architectures such as the LISP machine), the tags of arguments are automatically used to dispatch to the right routines by the microcode in generic arithmetic . . . . Microcoded machines typically can arrange for the tag field to be easily or automatically extracted upon memory fetch. Stack hardware can either have byte instructions suitable for tag extraction or can arrange for other field extraction, relying on shift or mask instructions in the worst case. Runtime management of types is one of the main attractions of microcoded LISP machines.
(Gabriel, 1985, pp 14-15)

(3) Garbage collection is an important architectural feature needed to support LISP software. LISP list creation, storage, and manipulation use large amounts of virtual memory. Since LISP pointers and their objects are allocated to a portion of memory, this LISP pointer and object space progressively become "garbage" as pointers and objects become inactive (i.e., unreferenced). Garbage collection is generally accomplished by following all active pointers in an old memory space. These LISP pointers and their LISP objects are copied to a new memory space leaving only garbage. This freed memory space is then reclaimed. Baker (1979) has reviewed the allocation of memory space in relation to garbage collection.

(4) The technical manuals for the LMI Lambda (List Machine, 1984), Symbolics 3600 series (1984), and Texas Instruments Explorer (1984) provide detailed technical information on AI architectures in support of LISP as well as some differences in this first generation of AI machines to grow out of the MIT CADR. The Xerox AI machine was developed out of an earlier MIT TX-2 machine. The Alto and the Xerox D-series AI machines had bit-mapped display systems and a mouse (a computer peripheral for manipulating screen images). These features were later incorporated into the other commercial AI machines developed out of the MIT CADR technology. Bit-mapped graphics are considered by some computer hardware traditionalists as a mixed blessing. These memory-heavy graphics require a vast amount of paging and virtual memory. The requirement for a machine to yield understanding as well as speed, costs machine memory and machine cycles. The Xerox AI machine and its development have been reviewed by Perry and Wallich (1985).

2-11. AI HARDWARE

a. The assessment of AI hardware was limited to dedicated AI systems with RAM memory greater than 1 Mbyte. The dedicated AI systems have wrapped their hardware around the conventions of the LISP language and tend to run LISP more effectively and efficiently than general purpose computer systems. The overview of symbolics LISP machine architecture by Moon (1987) generally supports the notion that specialized AI hardware is required to program large-scale AI applications software. Winston and

Brown (1979) suggested an "AND" logic gate metaphor from electrical engineering to illustrate LISP machine hardware. Although somewhat dated, the concept illustrated in Figure 2-7 provides a good illustration of the additional features required in processing symbols and relationships. In the figure, the first grouping refers to memory and memory management. The second group deals with microcode. The third group deals with using LISP as an implementation language above the microcode level. The fourth group deals with service (Winston & Brown, 1979, p 341). The requirements for a LISP machine are generally similar to those of general purpose computers:

> . . . these are but a few of many things that must be right to have a viable processor for LISP. The diagram in the figure is meant to suggest an AND gate--having most of the listed features does not do much good. Having all of them properly brought together is the source of the LISP machine's revolutionary power.
>
> We stress that these demands are not the idiosyncratic demands of a particular language. Rather, they are the demands always pressed by any language that supplies the data representation and manipulation features inherently required by intelligent information processing. We also stress that having these features does not make the LISP machine less capable at more traditional computing tasks. The demands of intelligent information processing are a suoerset of the demands of ordinary good computer design.
>
> (Winston & Brown, 1979, p 342)

Gabriel and Masinter (1982) and Gabriel (1985) reviewed LISP implementation on general purpose computer architectures and AI machines. Future AI architectures that are under development are reviewed by Mathews, Manuel, and Kruegar (1985, 1986). Experimental AI architectures are detailed by Schneck, Austin, Squires, Lehmann, Mizell, & Wallgren (1985), and Hillis (1985). Matthews et al. (1985) has suggested that many advances in AI architectures will be possible with custom-designed, very large-scale integration (VLSI) switch circuits:

> "VLSI is the key to future machines, so the question is how to use VLSI technology. There are three general approaches which can be taken: 1) creating uniprocessor architectures with internal parallelism, 2) creating an architecture to be a collection of specialized processors working cooperatively and 3) moving to massively parallel architectures".
>
> (Matthews, Manuel, & Krueger, 1985, p 1.2-7)

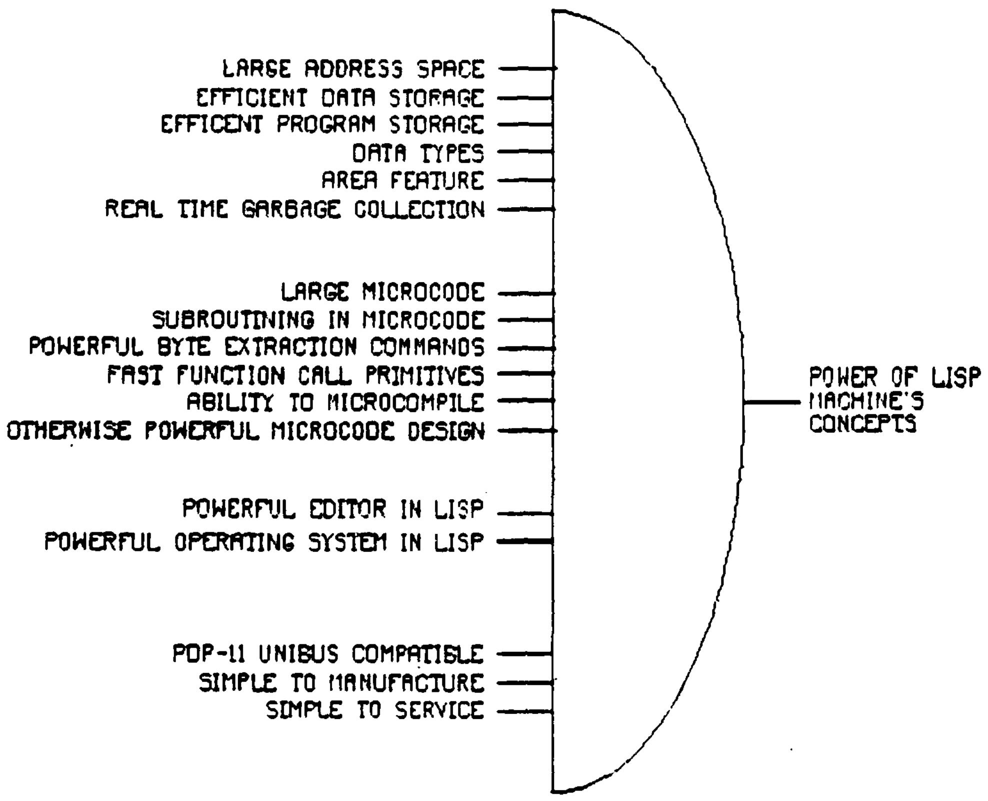

LARGE ADDRESS SPACE
EFFICIENT DATA STORAGE
EFFICENT PROGRAM STORAGE
DATA TYPES
AREA FEATURE
REAL TIME GARBAGE COLLECTION
LARGE MICROCODE
SUBROUTINING IN MICROCODE
POWERFUL BYTE EXTRACTION COMMANDS
FAST FUNCTION CALL PRIMITIVES
ABILITY TO MICROCOMPILE
OTHERWISE POWERFUL MICROCODE DESIGN
POWERFUL EDITOR IN LISP
POWERFUL OPERATING SYSTEM IN LISP
PDP-11 UNIBUS COMPATIBLE
SIMPLE TO MANUFACTURE
SIMPLE TO SERVICE
POWER OF LISP MACHINE'S CONCEPTS

Figure 2-7. Requirement for a powerful list-processing computer. The first grouping refers to memory and memory management. The second group deals with microcode. The third group deals with using LISP as an implementation language above the microcode level. The fourth with service. (Winston & Brown, 1979, p.341)

Figure 2-7. Requirements for a Powerful List Processing Computer

The design of megabit VLSI microchips is a complex task. Mead and Conway (1980) have suggested that design aids are needed in VLSI chip design. Chao (1983) has suggested the use of computer graphics for VLSI systems design. The use of expert system software may also be useful in designing VLSI chips used in multiprocessor system architectures. Kaisler (1986) has suggested that many AI applications will require parallel processing architectures. The latest and best work in AI architectures appears to suggest that computing human intellectual processes may require a great deal of raw computer power that is directed to its computing job with a great deal of precision. Brute force computing alone does not appear adequate to reproduce intelligent processes on a machine. Massively parallel computing architectures appear to give a great deal of computing power and manage complexity in a way similar to that of the human brain (McClelland and Rumelhart, 1986; Rumelhart and McClelland, 1986). It is unfortunate that in its present state of development, specific parallel processing AI architectures may be needed for each domain-dependent application. In the specialized domain of automated chess playing the use of VLSI technology has advanced very rapidly. Until recently computer chess was largely restricted to machines playing other machines. Within the last year (Berliner, 1986; Berliner and Ebeling, 1986), the leading machine has beaten four human chess masters in tournament games. The great power of this automated chess machine was achieved by wrapping the software into the VSLI hardware features:

> The SUPREM architecture is realized in Hitech, CMU's (Carnegie-Mellon University's) chess machine/program. The hardware can process approximately 200,000 chess positions per second. This includes generating a move, and incrementally maintaining all representations and the values that these generate, and retracting all these when the time comes to back-track in the search (this allows rapid movement through the search space with fast evaluation of the nodes in the knowledge representation)... It is in the method of move generation and evaluation where we differ from previous efforts. Our move generator is based on the parallel computation of all possible legal moves, of which there are only about 4,000. This means that there can be no more than 80 'ever-possible' moves by one side to a single square. The identification of which of these ever-possible moves is possible in the current position is performed on an array of 64 identical custom VLSI chips, one for each destination square. The fact that all moves are recognized simultaneously by different agents allows each agent to evaluate its a priori choice for best move to its square. It is then possible to choose among the best moves that each of the 64 chips tenders.
>
> (Berliner & Ebeling, 1986, pp 5-7)

b. Appendix D contains an AI hardware analysis for AI machines currently available. Figure 2-8 illustrates the relationship between AI hardware and general purpose machines. The notion of parallel matrix multiplication forms driving computer hardware development was stated by Charlesworth and Gustafson (1986) and was developed into a method to analyze new hardware by Cooper (1986):

> 11. The CAA 'Hardware Space' figure includes a 'Tendencies Key' to the special notation. That key is intended to suggest--
>
> a. n- 2 to 512. Coarse to mid-grained parallelism in processors and perhaps also memories.
>
> b. n- greater than or equal to 1000. Very fine-grained parallelism in processors and perhaps also memories.
>
> c. M- memory. A separately identifiable memory unit regardless of whether it is fairly small or very large.
>
> d. iP- fixed point processor. A processor definitely providing fixed point processing in hardware. It may perform floating point operations via software. The emphasis in symbolic and very fine-grained parallel processing tends toward iP. Much so-called scientific computing can be performed with iP's if the parallelism is fine-grained enough.
>
> e. fP- floating point processor. A processor definitely providing floating point processing in hardware. It may and usually does also provide fixed point processing in hardware (largely for addressing computation) but some parallel systems calculate memory addresses apart from the fP processors. (A famous classic computer, the CDC 6000 series provided hardware floating point multiplication but not fixed point multiplication.) The usual tendency is to think of scientific processing and fP as going hand-in-hand. But fP need is often more a result of aggregation in the sense of providing only coarse- to mid-grained parallelism. In the terms of a numerical analyst, needing or not needing fP is very much influenced by the mesh or grid size used to discretize a space and/or time continuum.
>
> f. v- vector(processor). Although there are different ways to perform vector processing, the tendency intended here is largely one of specialized pipelining rather than distributing elements to many essentially scalar processors. In particular, if a 'v' appears as a modifier within a parallel symbol in the figure, vi implies some sort of special vector device 'added on' on each processor node.
>
> 12. The composite symbols within the vertices of the figure use nesting of parentheses consistent with ordinary algebratic and English.
>
> 13. The '*' symbol represents multiple copies of the following expression (consistent with parentheses).
>
> 14. The composite symbol 'P,M' is to suggest that processor and memory are distinct. The symbol 'PM' is to suggest an architecture in which a processor and memory tend to blend into a single entity -- as in some of the massively parallel, very fine-grained systems.

A 3—D "Hardware Space" for Combat Analysis

[3/5/86]

<u>TENDENCIES KEY</u>

n -> 2 to 512	iP -> fixed point processor
nn -> ≥ 1000	fP -> floating point processor
M -> memory	v -> vector (processor)

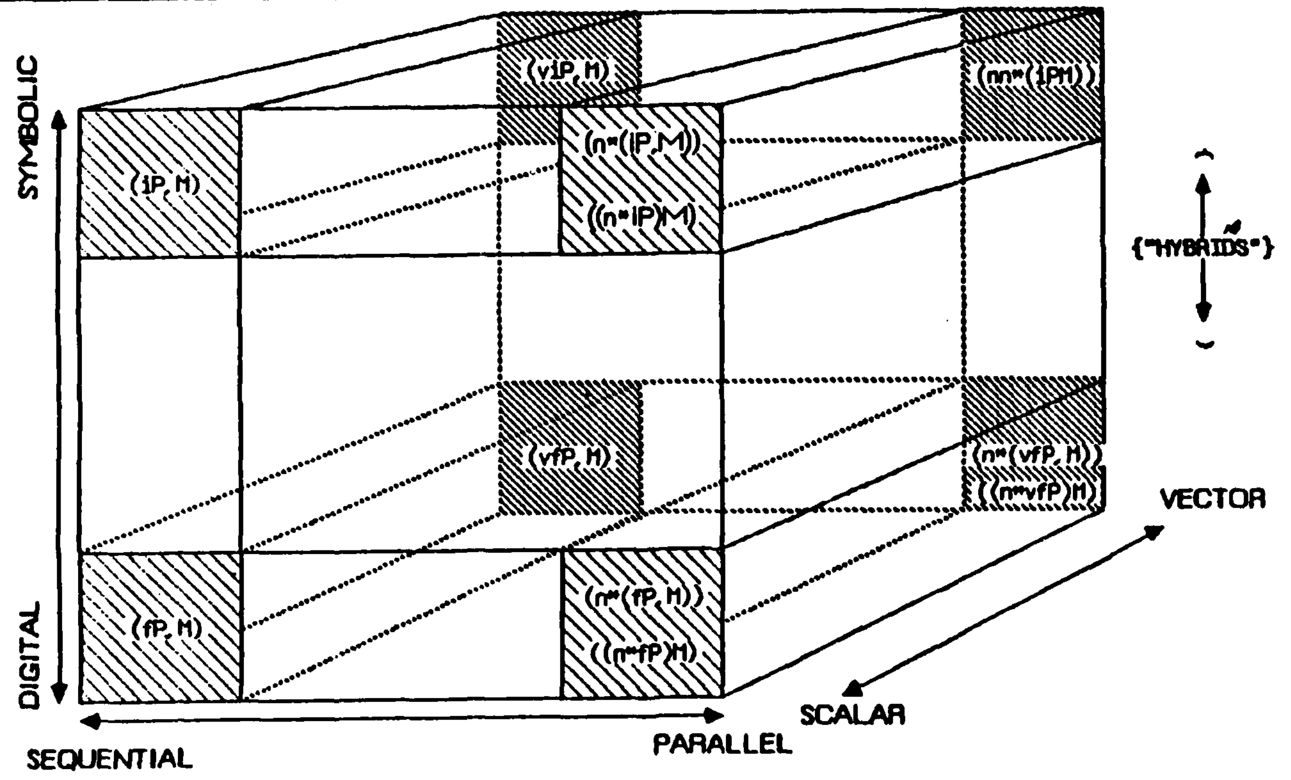


Figure 2-8. Symbolic and Numeric Computers for Sequential and Parallel Processing of Scalar and Vector Data

15. The distinction between local and shared memories is symbolized as --

(n*(P,M) local memory
((n*P)M) global memory

This distinction is one of the most crucial with regard to model formulation. The first kind of generic parallelism, (n*(P,M)), for practical purposes, forces an object-oriented, message passing formalism. It permits total memory 'n times' the address space of a single processor; the net practical address space may be reduced by requirements to clone copies of program and data across nodes. Software utility developers appear to be many years away from providing tools that can automatically object-orient an originally non-object-oriented (i.e., process-oriented) formulation. Very few analyst-programers appear to have mastered the art of writing original object-oriented models that slide directly into parallel achitectures.
16. The second kind of generic parallelism, ((n*P)M), forces all the processors to use the same address space. Some parts of the global memory may have to be reserved as local to each of the separate processors. Object-orientation and message-passing formalism are not required for this kind of parallelism. Formulation of large-scale models is difficult, but some analyst-programers believe that global parallelism makes model formulation inherently easier than for local parallelism.

(Cooper, 1986, pp 24-25)

At this stage of development, AI hardware needs to be used in conjunction with general purpose computer hardware in order to provide an environment that is useful for analysis. The notion of yolking symbolic and numeric hardware and their supporting software together is useful. There is a possibility that the resulting metacomputer can have an efficiency greater than the sum of its components.

c. AI hardware provides an optimal means to manipulate graphs and tree structures used in AI software. AI hardware has been optimized on symbolic processing and is therefore not an efficient medium for large-scale numeric applications. Although numeric applications have been shown to run efficiently on AI hardware (Gabriel, 1985), it can generally be concluded that AI is not appropriate for many general purpose applications. The development of AI software, toolkits, and applications programs appears to lag the introduction of new AI hardware by 24 to 60 months.

d. Table 2-6 illustrates the development of CAA's computer hardware from 1981 to 1985 (Hurd, 1985). The acquisition of computer hardware shows a smooth growth which appears to parallel the development of computer hardware technology.

Table 2-6. CAA Computer Systems Vital Statistics, 1981-1985

Sperry 1100/80 System	**1981**	**1985**
Number of CPUs	2	4
Memory (in mega-words)	2	4
Disk capacity (in mega-words)	708	2,645
Number of tape drives	7	8
Number of printers	5	4
Communications system	GCS	DCP
Number of active terminals	18	32*

* 36 additional ports available to support (100+) PCs
Current ports can also support additional terminals

Note: 1 Mega-word = 1,048,576 words = 4,194,304 characters (bytes)

VAX 11/780	**1983**	**1985**
Memory (in mega-bytes)	2	8
Disk capacity (in mega-bytes)	512	922
Number of terminals	6	14
Number of tape drives	1	1
Number of printers	1	1
RAMTEK graphics stations	3	7
Superset PGM Workstations	4 (4 PGM-1)	5 (3 PGM-1) (2 PGM-2)
TEKTRONIX Graphics Terminals (workstations)	2 (4014, 4081)	3 (4014, 2-4107)

Microcomputers	**1981**	**1985**
OSI CP-IVD	2	2
Apple II	1	1
IBM PC		3
IBM PC-XT		5
INM PC-AT		30
SYMBOLICS Workstations		
3670		1
3740		1

Section III. CAA MODELS, METHODOLOGIES, AND STUDIES

2-12. AN OVERVIEW OF CAA MODELS, METHODOLOGIES, AND STUDIES

a. **Overview.** Most analytic studies conducted at CAA use computer simulation models. Historical data on theater-level war are over 40 years old and do not reflect advances in technology and weaponry. Combat simulation models bridge the gap between historical records of war and expected effects of new techniques on theater-level battles. Combat simulation provides a tool to study the effects of varying doctrine, organization, and equipment mixes on theater-level combat. This section is limited to observations about theater level models and their applications outlined in Army Regulation 5-11 (US Army, 1983). The theater level models are part of a linked model hierarchy that includes models of corps/division and combat/support (i.e., battalion) task force levels of scope. The model hierarchy of computerized combat simulations is called the Army Model Improvement Program (AMIP). A variety of models at each force level are developed to serve as training, interactive (man/machine wargames), and automated simulations. These models evolve to adapt new technologies to the needs of the Army. The Army Models Committee (US Army, 1986) influences the models and the linkage between the models. The computer simulation model allows forecasting and decisionmaking to be made under controlled quantitative conditions. This has proven extremely useful for analysis and development of Army policy and doctrine. Model methodology allows for the systemic configuration, calibration, and control of the modeling process. Methodology controls the mathematical and nonmathematical logic that is implicitly and explicitly embedded in the combat simulation models. This section reviews models by their characteristics and purposes. The evaluation of the models will provide some insights into issues involved in modeling land combat. Models used by CAA during fiscal year 1985 were considered in this evaluation.

b. **General Characteristics of Combat Simulation Models.** The modeling of theater-level land combat is an abstract impressionistic enterprise (e.g., it contains no reference to weapon ranges within a line of sight). Tiede (1978) has suggested that modeling ground combat requires a level of difficulty surpassing air or sea combat modeling.

> The question of incorporating the effects of theater level decisionmaking . . . may be useful to extend the reasoning to this level for the sake of the completeness of the argument. The establishment of the initial conditions for the high-resolution combat models should reflect two different kinds of theater level decisions:
>
> - Policy decision for the engagement of extended range targets not included in the high-resolution model
> - Decisions that allocate resources to the combined arms forces modeled in the high-resolution model
>
> Models to implement policy decisions for engagement of extended-range targets are essentially tactical air campaign models . . The allocation of resources (to include missions and general plans for operation) to corps/divisions is a more complex

> problem than the engagement policy problem. This, too, is an optimization problem in that the theater commander is in fact establishing a campaign strategy for both air and ground forces. The degree to which this process is relative and dependent on information flow is not well understood at this time . . ..
>
> (Tiede, 1978, pp 160-161)

The resolution and complexity of the physics and terrain in land combat often exceed the limits of algorithmic representation. Three levels of combat models were suggested by Tiede (1978) to describe land warfare. Descriptions of land warfare are generally represented by: (1) controlled troop exercises; (2) wargaming; and (3) computer modeling. Due to limitations on size and time, CAA uses only wargames and computer modeling for its studies program on theater land warfare. Most wargaming at CAA is of the rigid type (i.e., descriptions of forces are represented by control elements and rule sets). Wargaming is also limited to computer assisted games used for studies of strategy and planning. The difficulty in training theater-level battle players (i.e., players that have the ability to abstractly conceptualize battles beyond the earth's horizon) limits the use of these models in studies at CAA. The model of choice for theater-level analytic studies at CAA is a fully automated (i.e., no human interaction with the model) computer model. Computer models used at CAA are classified according to the following characteristics:

(1) Acronym (popular identifier)

(2) Name

(3) Size

(4) Category/Interest Code

(a) A - Active

Suffix G - computer graphics program

Suffix M - model or multimodel system

Suffix S - support routine

Suffix U - utility routine

Note Gi - interactive graphic option(s)

Note Go - supplemental graphics option(s)

(b) C - Conversion/Test

(c) D - Development

(d) I - Interest (i.e., developed outside CAA)

(e) O - Outside (i.e., used by CAA outside Agency)

(f) S - Standby (i.e., available in-house)

Appendix E contains a detailed description of the models of interest to CAA. More comprehensive listings of models used in the area of land combat simulations can be found in the Deputy Under Secretary of the Army for Operations Research Catalog of War Games, Training Games, and Combat Simulations (US Army, 1983) and the Joint Chiefs of Staff Catalog of Wargaming and Military Simulation Models (Quattromani, 1982).

c. **Model Parameters.** Some of the informal descriptors of CAA models include physical measures such as lines of code, amount of core storage, and run time. For example, FORCEM consists of over 130,000 lines of code, takes 2 million Sperry 36-bit words, and runs over 24 hours on the Sperry 1100/84 computer. Because models at CAA are under constant modification, no strict description using these parameters is kept. Other model parameters such as sensitivity, validity, transportability, and usability (i.e., the ability to modify the model) are applied on an individual basis to the model by the user. Since most CAA models simulate the allocation of resources, they can be classified on the assumptions they make about the nature of allocation. Most CAA models assume a deterministic relationship between theater-level combat attrition and the allocation of resources. Stochastic assumptions of resource allocation incorporated in economic/cost modeling have been used to a lesser degree at CAA.

2-13. CAA MODELS IN FY 1985 BY PURPOSE. Twenty-seven models were used at CAA for studies conducted in fiscal year 1985. These models can be classified by purpose into five categories listed in Table 2-7. These categories are only generally descriptive of the CAA models. Because many of the CAA studies are bound into models, the study and the model often appear the same. Appendix E contains the complete names and descriptions of each model.

Table 2-7. CAA Models Used in FY 1985 by Function

I. Maintain Readiness (five models)

PRIM
E-DATE
NAPM
PSAM
PFAM

II. Direct Force Modernization (three models)

MICAF/AFP
RECPOM
ARQ

III. Mobilize and Deploy Forces (five models)

MOBREM
TRANSMO
SITAP
RIM
CAMP

IV. Conduct Tactical Warfare (nine models)

COSAGE
CEM
FORCEM
CFAS
FDM
TPM
ATCAL
PUTAR
NUFAM

V. Sustain Field Operations (five models)

FASTALS
TARMS
WARRAMP
CAS STRAT
PFM

2-14. CAA METHODOLOGIES. Methodologies are often used to compute values within a simulation model. Other times, a methodology is used either to aggregate values that are inputs to models or to disaggregate model output. Occasionally, a methodology is used to process data independently (as in a cost study). A survey of operations research tools and techniques conducted by CAA's Management Division in December 1985 yielded no consistent definition or pattern of use for methodologies. Many studies and models were cited as a tool or technique used at CAA during 1985 to accomplish studies. The following methodologies were cited as used by CAA in the performance of the study process in 1985:

(1) Combat Modeling

(2) Analog Modeling

(3) Statistical Analysis

(4) Mathematical Programing

(5) Queuing Theory

(6) Time Series Analysis

(7) Computer Assisted Wargaming

(8) Economic Analysis

(9) AI/Expert Systems

(10) Experimental Design

(11) Computer Programing

(12) Preparation and Presentation of Data

(13) Computer Simulation

(14) Superset Graphics

(15) Digital Modeling

(16) Political/Military Gaming Techniques

(17) Support Operations Modeling

(18) Cost/Benefit Analysis

(19) Bayesian Analysis (informal)

(20) Data Base Research

(21) Learning and Adaptive Systems

A re-analysis of the operations research techniques used in the fiscal year 1985/1986 Study Program yielded a more coherent picture. A management analysis of 115 studies yielded 10 methodologies that were distributed as follows:

(1) Political/Military Gaming (1 study)

(2) AI/Expert System (2 studies)

(3) Econometric Forecasting (2 Studies)

(4) Network Analysis (3 studies)

(5) Analytic Modeling (5 studies)

(6) Economic Analysis (7 studies)

(7) Computer Assisted Wargaming (9 studies)

(8) Mathematical Programing (19 studies)

(9) Combat Simulation (32 studies)

(10) Statistical Analysis (35 studies)

It appears that a clearer pattern of methodology use emerges over time. This pattern may be more of a function of the size and scope of the studies conducted at CAA than of the utility of any single methodology. Moglewer (1984) suggested another possible explanation for this widely disparate view of methodologies. It was suggested that the United States used inductive cognitive processes to approach operations research in defense planning and the operational art. Essentially, this approach attempts to describe, measure, explain, predict, and theorize in order to reach an induction. This approach is the converse to the Soviet approach of beginning with theory and proceeding to measurement in order to verify a deduction. The results of the CAA survey of methodologies appear to confirm this conclusion.

2-15. CAA STUDIES. Studies at CAA are conducted under the guidelines established in the Army Studies Program (Army Regulation 5-5). The essence of CAA study effort is to give the Army staff an internal analytic capability to study its policy and doctrine in relationship to external requirements such as Congressional legislation. CAA's study output is utilized by the Army in a similar manner to the Air Force's use of the Air Force Center for Studies and Analyses. During fiscal year 1985, CAA performed 49 studies. CAA studies that are beginning, ongoing, or will be completed in fiscal year 1986 can be grouped by purpose into the following five categories:

(1) Develop Planning, Programing, Budgeting, and Execution System inputs

(2) Analyze mobilization, deployment, and operational capabilities

(3) Design Army force alternatives

(4) Maintain force data bases

(5) Improve analytic techniques

A precise count of studies that take place in each category during any given year is difficult to obtain. Many studies begin in one fiscal year and overlap two fiscal years. Other types of studies combine several study efforts under one study. For example, the Light Helicopter Fleet Study is actually a family of studies. The general range of the CAA study effort is between 45 and 65 study completions per fiscal year. To gain insight into the CAA study program, a benchmark was created using Hughes' (1984) classification categories for models and the 49 studies completed at CAA during fiscal year 1985. The analysis of studies should give a snapshot of the study process (Table 2-8).

The analysis of the study process was based on a sample of studies conducted during two fiscal years in a peacetime environment. It is very probable that the proportion of studies devoted to any one category would change drastically during wartime.

Table 2-8. Fiscal Year 1985 CAA Studies by Model Category
(page 1 of 2 pages)

I. Battle Planning (24.5 percent)

1. Army Force Planning Data and Assumptions, FY 85-94
2. Measuring Improved Capability of Army Forces
3. Mid-Range Force Study - 84
4. Total Force Capability Assessment FY 90 Army Movement Requirements
5. Division Combat Sample Library
6. Force Level Analog Modeling Evaluation
7. Special Forces Requirements
8. Army Strategy Processing Model - Test I
9. Defense Planning Questionnaire - Army Mobilization Analysis
10. Nuclear and Chemical Assessment Data
11. Support Force Requirements - Northeast Asia
12. Army Force Planning Data and Assumptions, FY 86-95

II. Wartime Operations (0.0 percent)

III. Weapons Procurement (6.1 percent)

1. AH-64 Operational Availability/Inherent Availability
2. Maximizing CH-47 Corps/Division Flying Hours
3. Division Air Defense Special Study

IV. Force Sizing (16.3 percent)

1. Light Infantry Division Capability Analysis - Firepower and Survivability
2. Light Corps Capability Analysis - Corps Analysis
3. Light Corps Capability Analysis - Airborne Corps Firepower Potential
4. Measuring Improved Capabilities of Army Forces - Improved Allocation to Achieve a Capability Target
5. Measuring Improved Capability of Army Forces - POMCUS/PURE Excursion
6. MAXFLY - Planned Storage of Aircraft
7. Tank Distribution Analysis
8. TRASANA Aircraft

Table 2-8. Fiscal Year 1985 CAA Studies by Model Category (page 2 of 2 pages)

V. Human Resource Planning (20.4 percent)

1. Network Analysis Planning Model
2. Unit Replacement System Analysis
3. Wartime Manpower Planning System Casualty Estimates
4. Network Analysis Planning Model - Judge Advocate General Corps
5. Army Awards Analysis (Reserve Component)
6. Armor Resource Training Study
7. Casualty Replacement Rates
8. Wartime Manpower Planning System FY 87-91
9. Casualty Estimation for Contingencies
10. Evaluation of the Military Entrance Physical Strength Capacity Test

VI. Logistic Planning (18.4 percent)

1. Overview/PARCOM Turnkey Projects
2. Improved Methods of Automated Logistic Systems Development
3. Development of the Analytic Requirements Model
4. Analytic Assistance to Tank-Automotive Logistic Office
5. Transportation Model Comparison
6. Transportation Workload Forecasting Study - Implementation
7. Southwest Asia LOGCAP Estimate Study
8. Army Physical Inventory Control Procedures
9. Wartime Requirements for Ammunition and Materiel, Korea FY 90

VII. National Policy Analysis (10.2 percent)

1. Program Forces Capability Analysis - FY 90
2. Program Forces Capability Analysis - FY 92
3. Integrated Warfare Scenario Study
4. Joint Program Assessment Memorandum - Army Movement Requirements
5. OMNIBUS Capability Study

VIII. Command, Control, Communications, and Intelligence (0.0 percent)

IX. Cost (4.1 percent)

1. Management of Information Analysis
2. MICOM Industrial Fund Operations

2-16. CAA STUDY PROCESS, ISSUES, AND ANALYSIS. The CAA study process is designed to be responsive to study tasks formulated for near-term Army staff requirements as outlined under Army Regulation 5-5. Figure 2-9 illustrates the study process and its desired attributes (Vandiver, 1986). This paradigm represents a baseline notion of interaction of human expertise, computer hardware, and computer software in the CAA study process. It also provides a structure to integrate the effects of new technology and procedures into the CAA study process. An informal analysis of CAA studies using this paradigm of the CAA study process has indicated that a large portion of the entire study process is devoted to tasks that are outside the computer runs. The processes of preparing study directives, collecting and preparing input data for the study and the model, postmodel analysis, and study report generation rely on human expertise and appear to consume a great deal of study time. This appears to be a profitable area for the application of expert systems technology since it historically captures the study process in software for a future generation of analysts. Expert systems technology would also have the additional benefit of standardizing CAA study procedures. Networking expert systems software (i.e., distributed expert systems) would potentially speed the CAA study process by several orders of magnitude. The addition of symbolic processing capability to the model exercise portion of the process may add speed and flexibility to both the model runs and the overall study process by enabling the simulation to redirect itself during runtime. The coupling of symbolic architecture with numeric simulation engines would allow implementations of models that share files across a hierarchy of submodels. This would allow a customization of a new model to fit the requirements of the data and the study being conducted. An added benefit of coupling a hierarchy of models would be that selective data collection during runtime could be analyzed at different levels of resolution for different studies. The preceding notion of the applications of AI technology to the CAA study process and CAA models may have some generalizability to other study processes and other models within the Army Model Improvement Program (AMIP). AI technology, symbolic hardware and software, and the functional approach to modeling (i.e., the relations and behavior of objects within a simulation) may offer a dimension of speed and power to the analytic community. These attributes are not presently available and do not appear to be within the developmental limitations of current technology used by CAA or AMIP.

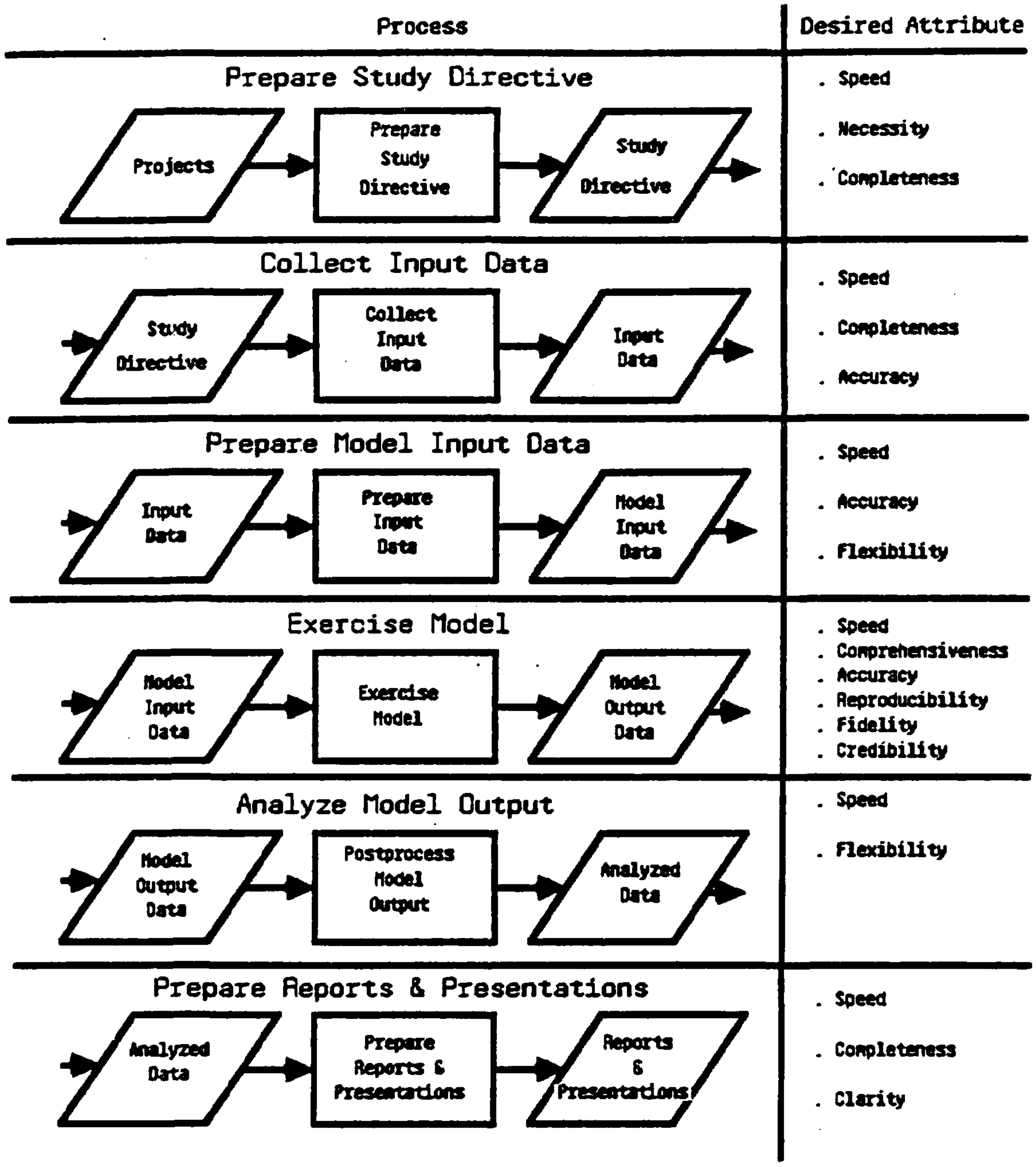


Figure 2-9. The CAA Study Process

Section IV. CAA TRAINING

2-17. CAA TRAINING

a. **Introduction.** CAA is a highly professional organization which trains toward excellence in performance. Appendix G provides an analysis and assessment of CAA skills during fiscal year 1985. The following assessment of CAA's training is provided to enhance the training effort toward excellence.

b. **Tasks for Training in AI.** Training in AI will tend to resemble the notions of the National Football League's training for championships. Team training is developed through management of individual skills in pursuit of organizational goals. Individual specialists need to be trained in terms of their task behaviors. Job task analysis is the critical variable in establishing individual training plans for user and programer level personnel. Education produces discriminate learning that focuses on producing task-related behaviors in individuals. Group training targets aggregate performance of individual members toward achievement or organizational goals. Feedback of individual performance must be inputed into a group. The analysis of this feedback is used to replan the training of the individual and the group. This process of training draws heavily on human factors research. Interdisciplinary teams such as those usually composing an AI effort must train as a group in order to achieve a coordinated effect. Training effectiveness analysis needs to be performed at CAA to analyze individual performance in term of the group or study team goals. Specialized training from AI teams requires the same care in planning and analysis that CAA normally uses in its study programs.

c. **Training for Excellence in AI.** Training CAA personnel in AI skills will require specialized training assistance from outside the Agency. The use of AI languages and skills is new to personnel at CAA. A few individuals in the Agency are generally acquainted with LISP and PROLOG from comparative programing courses in universities. No person within the Agency is skilled in building expert system software. As a minimum, AI software programers need training to perform:

(1) Programing of LISP/PROLOG

(2) Application of AI methodologies

(3) Application of expert system software

(4) Application of knowledge representation methodologies

(5) Application of automatic programing methodologies

(6) Creation and use of symbolic databases

The development of new skills at CAA will take an analysis of present skill levels within CAA and a plan for transfer of technology and training for new AI technology. In planning for AI training it is essential that the tasks required by the individuals within the Agency be separated from the tasks that must be achieved by the organization. In general, technology flows through organizations while technology transfer takes place through people. Dalton and Thompson (1986) have developed a four-stage model of professional career development which may be used to diagnose the organization and the individuals within it. The stages and their application to the organization are:

> In Stage 1, for instance, employees work under the direction of others as apprentices, helping and learning from one or more mentors. In Stage 2, they demonstrate their competence as independent contributors. In Stage 3, they broaden and act as idea leaders and mentors for others. In Stage 4, they provide direction for the organization.
>
> The four-stage model helps define what assignments are appropriate for each stage, and provides a framework for frank center discussion and training. For a company, the model can lead to professional development efforts and are less piecemeal. The organization can develop anticipatory programs rather than 'put out fires.'
>
> For employees, the model crystallizes what is expected of them at each stage. But more significant, it helps them recognize that ultimately they are responsible for their own careers. Organizations where career development efforts are linked to business needs are less likely to have these efforts cut off or ignored during 'bad times' such as the current slump in the computer industry. Thus the first step in implementing a professional development program is for management to determine the human resources needs of the company, by answering three major questions:
>
> - What human resources are needed to implement the organization's strategy or business plan?
> - What training and experience will ensure that the organization's strategy can be implemented effectively?
> - How critical to the organization is the retention of Stage 3 and Stage 4 employees (typically those over age 40), who really know how to use the system to make things happen?
>
> Once these questions have been addressed, the next step is to develop a profile of the workforce including such data as age, seniority, length of job assignments, educational level, and field of interest for each employee. Information on recent educational activities, like evening classes and company courses, are also useful, as are job performance ratings and salary.
>
> (Dalton & Thompson, 1986, p 43)

The skill levels for military officers and civilian workers are listed in Appendix G. This breakdown can provide a baseline for management to analyze current capabilities, experience, education, and training in planning for

AI technology transfer into CAA. Professional assistance to management from other Army agencies specializing in training development would be helpful in attempting to design training plans for AI technology for CAA.

d. Training for Transfer of AI Technology. The focus of a training program in AI can be directed toward using individual skills to transfer AI technology into the organization. A profitable area of focus, subject to verification by professional training analysts, would be to look at the following areas:

(1) Development of an in-house research staff in AI.

(2) Training/education of the application programers in AI.

(3) The use of New Equipment Training Officers (NETOs) from the Army Materiel Command to plan transfer of technology for new AI hardware, software, and methods.

(4) Conduct human engineering studies of man-in-the-loop in the CAA study process.

(5) Conduct a reverse engineering and fault analysis of CAA models and studies in order to feedback new refinements for future training into the training plan.

(6) Use Intelligent Computer-Based Instruction to teach LISP. Carnegie-Mellon University (Reiser, Anderson, & Farrell, 1985) has such a program. It includes:

(a) Audit trail of training.

(b) An error/fault analysis system which keeps track of systematic errors for groups of learners (i.e., buggy rules/bug catalogue) in order to improve the training system.

(c) An ideal student model (i.e., a simulation of the programing knowledge ideal students use in solving LISP problems).

(7) Use Los Alamos National Laboratories to train analysts in LISP and AI:

(a) The 17-week knowledge-based systems education course appears adequate to training application level AI programers in the theory, fundamental applications, and tools of AI.

(b) The training may need to be augmented by a further simulation training to meet the needs of CAA's models and studies mission.

(c) The projects required for the training course could be coordinated with organization goals identified in a prior training analysis.

(8) Use TRADOC schools for AI manager and programer training.

(a) An information paper (ATZH-CLC-A dated 28 October 1986 Subject: Army Artifical Intelligence Training Available at Fort Gordon) has indicated the availability of the following training during CY 1987:

<u>1.</u> Knowledge Engineering and Basic Expert Systems Building Course (2 weeks)

<u>2.</u> Advanced Course in AI Theory and Expert System Building (4 weeks)

<u>3.</u> Expert Systems Architecture and Advanced Skills in Object Oriented Programing Course (8 weeks)

<u>4.</u> Complete AI Training (14 weeks)

<u>5.</u> Introduction to AI in the Army (4 hours)

(b) The use of internal Army resources appears to be a very cost-effective alternative to training CAA application programers in AI techniques.

Section V. SUMMARY

2-18. SUMMARY. This chapter of the AI Study has provided a survey of AI software, toolkits, and hardware. The models, methodologies, studies, and training of the United States Army Concepts Analysis Agency were also surveyed. The description and analysis of these areas will provide the background and the basis to draw the conclusions and recommendations of the study.

CHAPTER 3

SUMMARY AND OBSERVATIONS

3-1. SUMMARY

a. The Artificial Intelligence Study (AIS) was conducted to meet the following objectives:

(1) Assess AI hardware, software, and toolkits that may be used to enhance CAA's models, methodologies, and studies.

(2) Assess CAA's current projects by study, model, and/or methodology.

(3) Assess CAA's skills, knowledges, and abilities to determine what type of AI equipment and training are most appropriate to accomplish CAA's mission.

(4) Propose ways to match AI technology with CAA models and studies to achieve near-term applications.

(5) Develop a small demonstration project which applies AI technology to CAA using tools which are currently available.

b. A survey of AI technology and CAA models, methodologies and studies was made. AI software was reviewed. LISP and PROLOG were reviewed in detail. AI simulation languages were also analyzed. Object-oriented programing for AI simulation was reviewed. A sample SIMSCRIPT program converted by Cassels (1985b, 1985d) into LISP was successfully run on a LISP machine. Knowledge representation methodologies were examined. Expert system software was surveyed for its applicability to CAA simulation models, and a comparison of the logic and use of AI software toolkits was made. Ten toolkits of a size and scope applicable to CAA's mission were reviewed.

c. A literature review of AI architecture and AI hardware that supports LISP was made. A three-dimensional notion to classify computer hardware was introduced. Other AI hardware and architecture were examined and analyzed. A summary of the hardware analysis is in Appendix D.

d. CAA models and methodologies were reviewed. General characteristics of combat simulation models and model parameters were used to describe CAA models and methodologies. The 27 models used by CAA for studies in fiscal year 1985 were categorized by purpose. A summary of CAA's models is in Appendix E.

e. The CAA study process was reviewed. The 49 studies conducted at CAA during fiscal year 1985 were classified by model category.

f. AI applications to CAA database management were analyzed. A recommendation for future directions was also given.

g. An analysis of training at CAA was made. The analysis relied on a skill assessment conducted during fiscal year 1985. A summary of the skill assessment is in Appendix G.

3-2. OBSERVATIONS

a. The following observations are reported for the AI study objectives:

(1) In response to study objective 1: assess AI hardware, software, and toolkits which may enhance CAA's models, methodologies, and studies.

(a) AI hardware, software, and toolkits provide an opportunity for the application of jump-ahead technology applications to CAA's models, methodologies, and study process. AI software may require specialized hardware for its development.

(b) The traditional notion that the available hardware defines the problem, models, and study capability of CAA is obsolete. User problems and software requirements may now be drivers for hardware acquisition and software development. The rapid prototyping of AI systems allows greater end-user involvement in model applications.

(c) The structure of AI hardware and software allows an audit trail of vendor, developer, and user logic to become embedded in the machinery. LISP-based models provide an audit trail which allows users and developers automated documentation and a graphic map of the model logic and action. Steele (1984) has suggested the use of the LISP property list for code documentation.

(d) Simulations based on VLSI chip technology may occur at great speeds. Because machine logic may be hardwired into silicon (e.g., VLSI chips), specific hardware like the TI compact LISP machine can be configured with VLSI chips to process great numbers of simulated military objects within theaters at high speeds.

(e) Several different types of numeric and symbolic computers may be coupled (yoked) together for modeling. The resulting metacomputer would appear to have a capability that is greater than the sum of its parts.

(f) The use of artificial intelligence will allow C3I systems within models to degrade the simulation of intelligent behavior that is modeled in steps. Rather than going from 100 percent to 0 percent in one step under increasing levels of battle stress, modeled C3I behavior can simulate human sensory overload behavior. The use of artificial intelligence in C3I logic in models to degrade modeled human performance gradually under stress allows a model to have a higher fidelity of simulation in the C3I structure.

(g) The user of fuzzy logic can replace two-valued syllogistic logic in combat simulation model logic:

from: (two-valued syllogism)--

all/no A's are B's
all/no B's are C's
all/no A's are C's

to: (fuzzy syllogism)--

some,few,many, 1-99 percent A's are B's
some,few,many, 1-99 percent B's are C's
some,few,many, 1-99 percent A's are C's
(Zadeh, 1985, p 417)

(h) It is possible and perhaps desirable to mix real-time simulation with sensor (i.e., situation reports) and database (i.e., AMSAA data) inputs into CAA models. The possibility to use mixes of simulated and sensor data in models may increase the fidelity of CAA simulations and allow theater-level models to become more useful for planning, problem solving, and the development and analysis of doctrine.

(i) Generic AI technology may be used to build rule-based/object-oriented models for multiprocessing architectures.

(j) The logic programing and logic machine technology being developed in Japan may be useful for CAA.

(k) AI technology can increase the utility and flexibility of theater-level simulation. Components of models may be fitted to the data, allowing customization of CAA models to sponsor requirements and new data.

(l) Decision support systems can use AI technology in simulations to support hypothetical constructs about future situations (i.e., ask "what if" questions).

(m) AI technology has applicability within CAA models in model representation, determination of variables for decisionmaking (threshold), battlefield representation, resolution (zooming), decision analysis, and real-time performance.

(n) AI technology may be used to evolve simulation from procedural representation of physical constructs toward declarative representation (objects). This, in effect, would link the physical representation of theater variables to decisionmaking using object simulation.

(o) The observed successory development of hardware, software, and practical applications typically requires overlapping intervals of 24 to 60 months.

(2) **In response to study objective 2:** assess CAA's current projects by study, model, and/or methodology.

(a) Problems of model improvement may be tied to the CAA study process.

(b) It is more useful to look at the process of studies than to analyze each individual study. The Vandiver (1986) notion of the CAA study process is an appropriate tool to assess CAA studies conducted during FY 1985.

(c) CAA sponsor requirements drive changes in studies, models, and methodologies; therefore, it is important to consider the sponsor when adding AI technology to CAA's studies, models, and methodologies.

(3) **In response to study objective 3:** assess CAA's skills, knowledges, and abilities to determine what type of AI equipment and training are most appropriate to accomplish CAA's mission.

(a) Technology transfer requires attention to people variables first. Transfer of AI hardware, software, toolkits, and methodologies requires an organization commitment to integrated individual and team training.

(b) AI machinery and software allow the complexity of problems in model building to be shunted away from the human being and into the computer. This notion of man-machine systems also allows for human judgment to become historically embedded inside the machine for future use.

(c) The construction versus the use of expert system software requires two different types of abilities. The use of expert system software with very friendly user interfaces is easy and requires no programing skills other than the ability to think clearly. The development of expert system software is difficult, time-consuming, and expensive. It requires many resources applied with the best plans of management.

(d) Human factors are critical in the development and transfer of AI technology into CAA man-in-the-loop (man-machine), workspace analysis for workstations, training, and training effectiveness analysis.

(e) Expert systems may be used for: job analysis, organizational/project design, customizing studies around personnel capabilities, and aiding nonprogramers to judge model validity.

(4) **In response to study objective 4:** propose ways to match AI technology with CAA models and studies to achieve near-term applications.

(a) An effective match for near-term applications between AI technology and CAA models and studies can be summarized using the Vandiver (1986) notion of the CAA study process and its desired attributes.

(b) AI hardware and software may be used to increase the speed and effectiveness of the preparation of study directives and preparation of reports and presentations. An intelligent study director's workstation could be developed or adapted from the CIA analyst workstation. The application of expert system software and intelligent study director workstations (AI workstations with graphic digitizer and datafile inputs, interactive graphics, and a laser printer) would provide a qualitative and quantitative improvement of the CAA study process. Intelligent workstations would aid the productivity of study teams.

(c) The use of symbolic graphical systems offers opportunities to improve all aspects of the modeling and studies process. For example, it is now possible to visually track the formatting, input, and flow of data through a model in real time. Emphasis should be given to graphic and expert system applications for preprocessing of models and the analysis of outputs. Graphics may also aid the efficient preparation of study directives and study reports. Graphics should be emphasized as a high payoff, quick return on investment technology.

(d) Systems of networked expert systems software may allow intelligent control of preprocessing, model exercise, and postprocessing of simulations. These types of software would also allow intelligent and efficient formulation of study requirements in terms of the capacity of the data and the models available at CAA and the analysis of model outputs. The networking of expert systems to control the entire CAA study process is possible. This network of expert systems would aid human intellectual efforts at CAA. Given a problem and an available data set, a possible extension of expert systems would be to adapt parts of models to the given problem/data context. Development of this network would require a large-scale research effort.

(e) It is possible for expert systems to mechanize the best military judgment, experience, and reasoning within model logic. The use of machine intelligence to explain reasoning from new facts and data can allow accumulated military knowledge to control doctrine development, decisionmaking, decision analysis, and training.

(f) Expert system software may be used to develop simulation software. Expert system software may be a necessary but not sufficient condition for structuring parallel processing software and hardware. The capacity for human decisionmaking becomes overloaded beyond seven to nine alternatives. Expert system software could ease the cognitive workload involved in developing algorithms and programs for parallel computers.

(g) The use of LISP machines for general software development would allow an increase of programer productivity. The use of LISP machines would allow rapid prototyping of conventional software that runs on traditional computers. The programing environment of LISP-based models would allow a programer to rapidly prototype incremental development of large models by using the interpretive features of the LISP language. The interlanguage calling capability of the CAA LISP machine allows access to CAA simulation languages (i.e., FORTRAN and maybe SIMSCRIPT). The tools for program development that are currently available on the LISP machine such as the symbolic debugger, incremental compiler and dynamic linker may speed-up program development by an order of magnitude. The use of templates in coding allows fewer coding errors to occur unnoticed. This notion of program development is similar to the assertion by Barr and Feigenbaum (1982) that AI programing languages have highly developed programing environments:

> To sum up briefly, AI programmers must impose some workable organization upon a large set of interacting modules, one that is flexible enough to allow constant modification, correction, and growth of the system. The support provided by a good programming environment is essential.
>
> In looking further at the question of why AI languages tend to have highly developed environments, it is also significant that the programming-support system of a language often resembles an AI system in its own right. It may rely on a large database describing the program and consist of several modules (editor, debugger, etc) that interact strongly. Consider, then, the advantages of developing environment facilities in an AI language, presumably the one being supported.
>
> Although one's wish list of programming-support features might be endless, the following list includes the most important ones:
>
> 1. An interactive language, that is, one in which statements can be typed in as commands and are executed immediately. Compiler-based languages are generally not interactive.
> 2. A good editor, if possible, one that can deal with a program according to its structure as a program (not just as text composed of characters).
> 3. Interactive debugging facilities, that is, breaks, backtraces, and facilities for examining and changing program variables.
> 4. Input/output routines. The most common input/output actions should be specially supported by standard system input/output functions, so that the programmer is not burdened with such details.
>
> (Barr & Feigenbaum, 1982, p 66)

(h) LISP-based model development concepts may be useful in CAA's model development and application efforts. Model development in LISP is possible for ADA or another higher order language (HOL) by using interpreter/translator or automatic programing facilities. A LISP to SIMSCRIPT interpreter/translator may be immediately useful.

(i) AI technology will require continuous updating of the management plan in order to keep the application of new technology current with new developments. CAA must accept some risks in adopting high technology to achieve jump-ahead advances in simulation, the study process, and operations research.

(5) In response to study objective 5: develop a small demonstration project which applies AI technology to CAA material using tools which are currently available or will be acquired in the next 60 days from 1 August 1985.

(a) The demonstration project using the Knowledge Engineering Environment (KEE) software toolkit on the symbolics LISP machine has shown that CAA's FORCEM command and control model logic can be duplicated in an AI environment.

(b) Expert system software does not replace intelligence but multiplies intellectual processes with the power of machinery. This effect is similar to a bicycle multiplying human kinetic energy.

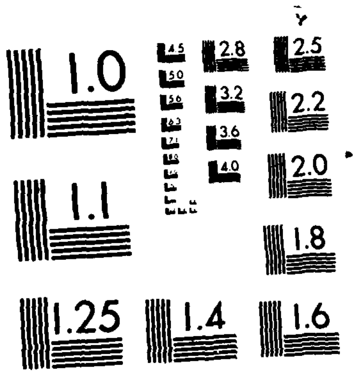
3-3. APPLICATIONS OF OBSERVATIONS. The following applications are identified for implementation of the observations:

a. The application of AI technology to the preparation of reports and presentations should be given first priority by CAA. This area of application is the most cost effective, has the lowest risk, and offers the highest payoff. The planned acquisition of the Xerox Star and a laser-graphics printer for the Publications Center should be accelerated by the Agency. An adaptation of the Golden Tiger Workstation (Xerox 1186 AI machine with a Notecards Toolkit, SMALLTALK-80 style LISP, and desktop publishing capabilities) from the current configuration used at the Central Intelligence Agency (CIA) to that required by study directors at CAA should be considered. One or more intelligent study director workstations could be given to each study director and networked to the CAA mainframe computer and the Publications Center. A separate study of the effectiveness of the analyst workstation at the CIA should be considered. The study could include the requirements needed to adapt the CIA analyst workstation to the requirements of CAA study directors. Based on the results of the study, a contract through DARPA should be considered to provide Engineering Applications Research (DOD Research Category 6.4) for adapting the software, toolkits, and hardware used in the CIA analyst workstation to the needs of the report generation, study presentation, and study directive preparation processes at CAA. The contract could include networking the study directors' workstations to existing and anticipated CAA computer hardware.

b. An interrelated series of expert system software programs [illegible] written for collection of input data, preparation of input [illegible] analysis of output for CAA models. This software would [illegible] and efficiency of the CAA models. An additional feature [illegible] would be to provide an audit trail for the CAA study [illegible] models.

AD-A181 029 ARTIFICIAL INTELLIGENCE STUDY (AIS)(U) ARMY CONCEPTS ANALYSIS AGENCY BETHESDA MD R B MODJESKI FEB 87 CAA-RP-87-1 2/3

UNCLASSIFIED F/G 15/6 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

The codification of the judgments used in model application could also be used to train new analysts and provide a historical record of the process for model improvements.

c. AI technology can be used to upgrade the judgment and expertise of the command and control logic in CAA theater-level models. The present logic of command and control is that of the analyst coding the model. The use of an AI toolkit may duplicate the existing model logic of FORCEM. The expertise and judgment of senior Army officers can be captured through the use of experimental design in a wargaming environment that is an analytic surrogate of FORCEM (i.e., FORGE). This expertise can then be transferred into the model systematically with the LISP-based toolkit. A CAA study in the use of intelligent command and control augmentations to FORCEM using the Knowledge Engineering Environment (KEE) toolkit and the Symbolics LISP machine should be considered.

d. Training of personnel is the key variable in the transfer of AI technology. A TRADOC training plan for the adaptation and use of AI technology at CAA should be considered. Army New Equipment Training Officers (GS-1710 series) could advise CAA on the proper instructional and training requirements needed for all new AI hardware, software, and toolkits.

e. The integration of AI systems and software with existing CAA hardware, software, and databases is possible and desirable. The development of a SIMSCRIPT interpreter/translator using symbolic programing tools would be a logical first step in this direction. This would enable application programers to debug and improve existing code with the power of the LISP machine.

f. The symbolic LISP machine should be used in new software development. This hardware provides the fastest development environment in existence for the prototyping, coding, and debugging of higher order languages. CAA application programers should be trained in using the LISP machine to develop programs in FORTRAN, SIMSCRIPT, COBOL, and other higher order languages used at CAA.

g. The application of AI graphics appears to hold great promise to improve the speed and effectiveness of the CAA studies and modeling process. A concerted effort to integrate AI graphics into CAA should be made, especially interactive graphics applications developed for simulations (i.e., Xerox LOOPS, Intellicorp's SIMKIT). The visualization of data as it is input into a node, is processed by the model, and is formatted as output may be extremely useful for model users and developers. A prototype effort in visual programing of CAA data flow through a theater-level model is an appropriate area for research for program application and the debugging of model code.

h. The use of AI technology in programing parallel machines should be explored. The use of expert system software may be able to reduce the complexity in programing and debugging theater-level software. Any exploratory efforts in parallel machines should include the use of AI technology in software development and maintenance.

i. AI technology should be used to analyze CAA models and studies. Information on the software reliability of CAA models can be studied with expert system software. A cumulative record of software reliability and study fault analysis should be kept within the Agency. Historical trends of software and studies can be analyzed to suggest improvement and improve quality control. Expert system software can use a simple rule-based system to suggest model revisions and improve the formulation of study directives to the sponsor's immediate and long-term organization requirements (i.e., cumulative studies by a sponsor may be used to shape the direction of future studies). A knowledge-based system of CAA studies and model use may be useful to analyze the study process.

j. An effort to modernize CAA database management with AI technology should be started. The use of expert systems software, AI toolkits, and LISP machines to assist structuring databases for studies and models can improve the speed and effectiveness of this process. The use of intelligent front-end software to allow browsing databases is useful to analysts for scoping the effort required to use a model for a study. A separate research effort needs to be made in AI applications for database management.

k. The CAA Advanced Research Projects Office (ARPO) should pursue short-term applications research and long-range research. This would exploit most aspects of AI developments.

l. An effort to monitor Japan's development in fifth generation technology in the literature should be considered. The Japanese efforts in PROLOG and PROLOG machines could be followed.

m. A monitoring of and adaptation of DARPA research in the Army's AirLand Battle Management Program should be considered for CAA use.

n. The integration of the CIA world database into CAA models should be considered. The world database provides a worldwide terrain map. Intelligence data gathered under the assumptions of the world database and added to CAA terrain maps in CAA models could avoid a source of systematic error. The CIA world database is based on AI technology. It could provide a good entry point to integrate AI technology into the CAA models. Its use could allow congruent intelligence estimates (i.e., estimates using the same assumptions) to be entered into CAA models.

o. An optimal configuration of numeric and symbolic hardware is desirable. Research may be conducted into the optimal mix of existing hardware for CAA data and software needs. Yoked designs appear possible and desirable.

p. In any contract development of AI needs, the specification of performance of AI systems and the addition of AI components to systems should be considered carefully. The development of the RSAC, for example, specified AI be used in the development of the model. The programing of RSAC in C language allowed the use of pointers and tables in the decisionmaking algorithms. The final product lacked a trackable explanation facility for decisions in the model that LISP would have provided. The development of future models at CAA should consider the cost/benefit and risks associated with AI based simulation models of land combat.

q. The concept of building from specification is optimal in an environment when the problem is well understood. When the problem is ill-defined, an incremental approach to model development and problem analysis may be fruitful. The application of AI technology to combat simulation models would require years of incremental development. The following paradigm could be used:

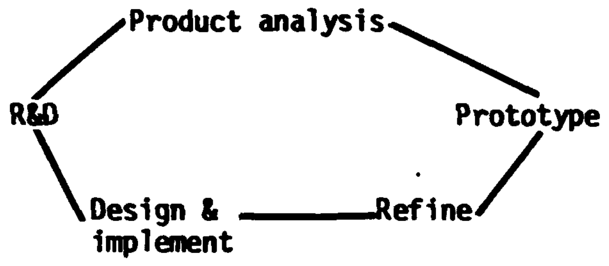


Figure 3-1. AI Technology Transfer Paradigm

This approach would allow a flexible development of models to occur. It would allow for the inclusion of new technology and the addition of new missions into the CAA studies and analysis programs.

r. Command and control applications rely heavily on judgment and doctrine. This appears to be an excellent area to begin a development and demonstration project of AI technology. Within this area there is an excellent opportunity for AI technology application in the following subareas: planning, communications, reaction (force deployment), models of human cognition for C2, and interpretation of enemy activities (i.e., recognizing situations before they occur, or determination of what the enemy can do in the future based on his current activities).

s. AI technology can be used to analyze non-AI models and studies. The use of an expert system to analyze patterns of software errors is possible. Just giving present CAA programers more power (i.e., more machine memory and cycle time) may not improve the quality of CAA's models. An expert system to document, trace, and analyze software failure at CAA may be needed. The determination of weak points in code would allow an incremental improvement of software. A starting point in this effort could be the detailed analysis of CAA software reliability. If software is unreliable, it is probably in error.

t. Fuzzy set theory will allow simulation modeling logic to handle a greater level of uncertainty. It will also allow relating statements of probability (e.g., PKs) and expected value to uncertainty. The use of fuzzy logic to understand the relationship of certainty to uncertainty may be valuable in itself for the modeling community. Fuzzy set theory may allow the modeler to deal with noise in the data and combine intelligence about data with the data itself (i.e., merge information). The possibility exists to use fuzzy reasoning techniques for control of deductions about data. Intelligent front-ended, database systems or preprocessors can merge and add uncertainty values to data and other model inputs.

u. A LISP-based FORCEM would allow systematic changes in the model to be made in an orderly manner. FORCEM, like many large simulation systems, is composed of many procedures that call each other in a complex way that becomes more difficult to trace as the system grows. A modification of procedure "A," for example, interferes with the optimal function of procedure "B" when it is called by procedure "C." A LISP-based FORCEM would allow programers and program development teams to understand the interaction of all FORCEM's pieces as they interact. LISP may provide a simulation support system for documenting and understanding changes to FORCEM.

3-4. DISCUSSION. The AI Study focused on AI technology and the process of combat modeling and studies. This approach was taken for several reasons. First, the distinction between AI software and AI hardware was blurred. For example, the notion of the Symbolics LISP machine having the LISP software pointers physically implemented in the hardware system made nonoverlapping categorization difficult. The concept of wrapping a specific problem around hardware and software applications appeared less desirable than designing applications around CAA processes. Second, the specific studies and analyses performed by CAA reflected changes in the mission of the Army. Many of the major studies were continuous and were only divided at a fiscal year for bookkeeping purposes. The most efficient overview of the process appeared to be Vandiver's (1986) view of the CAA study process. The following areas of application for the observations are summarized from paragraph 3-3 and compared against the Vandiver notion of the study process for long-term (greater than 60 months), medium-range (36-60 months), and short-term (12-36 months) applications (Table 3-1).

Table 3-1. AI Technologies and the CAA Study Process for Long, Medium, and Short-term Application

Observations	Prepare study directive	Collect input data	Prepare model input data	Exercise model	Analyze model output	Prepare reports/ presentations
1. Apply AI technology to preparation of study reports and presentation	S					S
2. Develop networked expert systems for I/O data from models		M	M		M	
3. Apply AI technology to upgrade C^2 logic in its models				S		
4. Use TRADOC for AI training (human factors)	S	S	S	S	S	S
5. Develop LISP/SIMSCRIPT interpreter for rapid program development				M		
6. Use LISP machines for software development				M		
7. Develop AI graphics applications	S	S	M	L	M	S
8. Explore parallel AI technology			M	M	M	
9. Explore AI technology for database management			M			
10. Update FORCEM code with LISP/SIMSCRIPT translator/interpreter				M		
11. APRO continue short- and long-term research	S	S	S	M	M	S
12. Monitor Japanese PROLOG/PROLOG machine		L	L	L	M	
13. Monitor DARPA efforts in AirLand Battle management model				M		
14. Adapt CIA world data bank I and II for model terrain inputs			M			
15. Research yoked symbolic and numeric hardware and software		M	S	L	S	

APPENDIX A

(U) STUDY CONTRIBUTORS

1. **STUDY DIRECTOR**

   Dr. Richard B. Modjeski, Research and Analysis Directorate

2. **TECHNICAL REVIEW TEAM**

   Mr. Gerald E. Cooper
   Mr. Robert Orlov
   MAJ Tommy E. Shook

3. **PRODUCT REVIEW BOARD**

   Mr. Daniel J. Shedlowski, Chairman
   Mr. James J. Connelly
   Dr. Andrew L. Ressler (Outside Member: DA AI Center)

4. ***OUTSIDE REVIEW OF OBSERVATIONS***

   Prof. Patrick H. Winston Ph.D.
   Director, Artificial Intelligence Laboratory,
   Massachusetts Institute of Technology

APPENDIX B

STUDY DIRECTIVE

DEPARTMENT OF THE ARMY
US ARMY CONCEPTS ANALYSIS AGENCY
8120 WOODMONT AVENUE
BETHESDA, MARYLAND 20814-2797

REPLY TO
ATTENTION OF:

CSCA-AST

19 SEP 1985

MEMORANDUM FOR ASSISTANT DIRECTOR, AS

SUBJECT: CAA Research into Artificial Intelligence Study

1. **PURPOSE.** This directive provides tasking for a study to adapt artificial intelligence (AI) and expert systems technology to theater-level force modeling.

2. **BACKGROUND**

a. The computer simulation models at the U.S. Army Concepts Analysis Agency (CAA) have been developed using higher order languages. Recent AI technology points to the inclusion of object-oriented programing and rule-based systems in computer simulation programs.

b. Representing knowledge of specialized fields of expertise may aid in the analysis of complex simulation models. CAA's work in the development and analysis of models and the analysis of model outputs may be aided by AI technology. Portions of simulation models, such as command and control (C^2), may be enhanced by using expert systems techniques.

c. AI methods provide a means of developing systems of knowledge and inference strategies to simulate a military expert's behavior within limited domains. For example, an expert advisory system could assist analysts in structuring the input in the COSAGE model. Some aspects of AI technology could assist in making existing computer simulation models more user friendly. A demonstration involving AI capabilities within current CAA work will provide insight into how AI technology may be applied at CAA.

3. **STUDY SPONSOR.** U.S. Army Concepts Analysis Agency.

4. **STUDY AGENCY.** U.S. Army Concepts Analysis Agency.

5. **TERMS OF REFERENCE**

a. Scope. This study will assess hardware and software requirements needed to use AI and expert system technology in accomplishing CAA's current mission.

b. Objectives:

(1) Assess AI hardware, software, and tool kits that may be used to enhance CAA's models, methodologies, and studies.

CSCA-AST
SUBJECT: CAA Research Into Artificial Intelligence Study

(2) Assess CAA's current projects by study, model and/or methodology.

(3) Assess CAA skills, knowledge and abilities to determine what type of AI equipment and training are most appropriate to accomplish CAA's mission.

(4) Propose ways to match AI technology with CAA models and studies to achieve near term applications.

(5) Develop a small demonstration project which applies AI technology to CAA material using tools which are currently available or will be acquired in the next 60 days.

c. Assumptions:

(1) AI/expert system technology may be applicable to military modeling.

(2) Rule based solutions may be implemented to support computer simulation models used at CAA.

d. Essential Elements of Analysis:

(1) Do AI applications and technologies currently exist which may be used in CAA's models or studies?

(2) Which of CAA's models or methodologies offer the highest potential for an initial AI application?

(3) Which hardware/software and AI tools are available and suitable for application to CAA's modeling and study effort?

6. RESPONSIBILITIES

a. Analysis Support Directorate (AS) has primary responsibility for accomplishment of the objectives listed in paragraph 5b. It will provide a full time study director.

b. Individual CAA directorates will be invited to participate and furnish information as required; study team members will be selected on an Agency-wide basis after consultation with the applicable Assistant Director.

7. LITERATURE SEARCH. A literature search of AI hardware, software, and application tools will be conducted. AI hardware, supporting software, and tool kits will be analyzed for the following systems and others which may become known during the study:

(1) Fujitsu
(2) IBM (PC)
(3) LMI

CSCA-AST
SUBJECT: CAA Research Into Artificial Intelligence Study

(4) Symbolics
(5) Texas Instruments
(6) Xerox

8. **PROJECT MANAGEMENT RESOURCES**

a. An estimated effort of 1.0 professional staff year will be required for this study.

b. Some travel and training will be necessary to develop the skills required to build an expert system prototype.

9. **Administration**

a. Milestones. (See attached Study Scheduling Report)

b. Products.

(1) A study report relating AI applications to CAA. An assessment of AI capabilities, CAA's capabilities to do AI work, and the type of work which CAA does to which AI technology is applicable will be developed in detail.

(2) A demonstration of CAA work using AI technology which uses existing equipment and software.

[illegible]

Encl

E. B. VANDIVER III
Director

STUDY TITLE
ARTIFICIAL INTELLIGENCE STUDY

STUDY DIRECTOR
DR. RICHARD MODJESKI

SUBMISSION DATA	INITIAL	REVISED
	851001	

START DATE	COMPLETION DATE *
850820	861031

EVENT/ACTIVITY	START	END
DRAFT AI HARDWARE ASSESS PLANS	850918	851130
DRAFT REVIEW OF CAA PROJECTS	850918	851231
DRAFT PLAN OF CAA SKILL ASSESS	851104	860115
DEMONSTRATION OF AI PROTOTYPES	851004	851126
STUDY REPORT: AI HARDWARE	860131	860331
STUDY REPORT: AI TECHNOLOGIES	860113	860331
FINAL ARB	860416	860416
PRB REVIEW	861006	861014
REPORT PUBLISHED	861031	870228

APPENDIX C

REFERENCES

Abelson, H. & Sussman, G. J. (1985). Structure and interpretation of computer programs. Cambridge, MA: MIT Press.

Army Science Board (1982). Report of Army Science Board ad hoc subgroup on artificial intelligence and robotics. Washington, DC: Office of the Assistant Secretary of the Army for Research, Development and Acquisition. (DTIC No. AD A122414)

Baker, H. G., Jr. (1979). Optimizing allocation and garbage collection of spaces. In Winston, P. H. & Brown, R. H. (Eds), Artificial intelligence: An MIT perspective, Vol. 2, Understanding vision, manipulation, computer design, symbol manipulation (pp 339-346). Cambridge, MA: MIT Press.

Barr, A. & Feigenbaum, E. A. (1981). The handbook of artificial intelligence (Vol. 1). Los Altos, CA: William Kaufmann Press.

Barr, A. & Feigenbaum, E. A. (1982). The handbook of artificial intelligence (Vol. 2). Los Altos, CA: William Kaufmann Press.

Bawden, A., Greenblatt, R., Holloway, J., Knight, T., Moon, D., & Weinreb, D. (1979). The LISP machine. In P. H. Winston & R. H. Brown (Eds.), Artificial intelligence: An MIT perspective, Vol. 2, Understanding vision, manipulation, computer design, symbol manipulation (pp 347-373). Cambridge, MA: MIT Press.

Bellman, R. (1978). Artificial intelligence. Boston, MA: Boyd and Fraser Publishing Company.

Berliner, H. (1986). Hitech wins North American computer chess championship. AI Magazine, 6(4), 30.

Berliner, H. & Ebeling, C. (1986). The SUPREM architecture: A new intelligent paradigm. Artificial Intelligence, 28(1), 3-8.

Bobrow, D. G. & Stefik, M. (1977). The LOOPS manual (XEROX PARC Research Report KB-VLSI-81-13). Palo Alto, CA: Knowledge Systems Area, XEROX Palo Alto Research Center.

Bobrow, D. G. & Winograd, T. (1977). Experience with KRL-0, one cycle of knowledge representation language. Proceedings of the International Joint Conference on Artificial Intelligence (pp 213-222). Cambridge, MA: Morgan Kaufmann.

Bobrow, D. G. & Winograd, T. (1985). An overview of KRL, a knowledge representation language. In R. J. Brachman & H. J. Levesque (Eds.), Readings in knowledge representation (pp 263-285). Los Altos, CA: Morgan Kaufmann.

Bobrow, D. G. (Ed.) (1985). Qualitative reasoning about physical systems. Cambridge, MA: MIT Press.

Bobrow, D. G. (Ed.) (1986). Qualitative reasoning about physical systems. Cambridge, MA: MIT Press.

Bogdanowicz, J., Tucker, G., & Sugden, E. (1986). Strategic computing for image understanding program: algorithm development on the Butterfly. Proceeding of the Butterfly/WARP User's Group Workshop February 18-19, 1986. McLean, VA. The BDM Corporation Westbranch Conference Center.

Bonasso, R. P. (1981). ANALYST: An "expert" system for processing sensor returns. Proceeding of the 1st US Army Conference on "Knowledge Based Systems for C3I (pp 221-245). Ft Leavenworth, KS: US Army Model Management Office.

Bonasso, R. P. (1983). Expert systems for intelligence fusion. Proceedings of the Army Conference on Application of Artificial Intelligence to Battlefield Information Management (pp 101-116). White Oak, MD: US Naval Surface Weapons Center, US Army Research Office.

Bourne, L. E. Jr., Ekstrand, B. R., & Dominowski, R. L. (1971). The pyschology of thinking. Englewood Cliffs, NJ: Prentice-Hall.

Branson, R. K., Rainer, G. T., Cox, J. L., Furman, J. P., King, F. J., & Hannum, W. H. (1975). Interservice procedure for instructional systems development, executive summary. Ft Benning, GA: US Army Combat Arms Training Board. (NTIS No. ADA 019486)

Brownston, L., Farrell, R., Kant, E., & Martin, N. (1985). Programming expert systems in OPS 5. Reading, MA: Addison-Wesley Publishing Co.

Bryan, G. L. & Regan, J. J. (1972). Training system design. In H. P. Van Cott & R. G. Kinkade (Eds.), Human engineering guide to equipment design (2d ed.) (pp 663-666). Washington, DC: American Institutes for Research.

Card, S. K., Morgan, T. P., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cassels, R. A. (1985a). SIMSCRIPT job-shop. LISP.6 (computer program). Cambridge, MA: Symbolics, Inc.

Cassels, R. A. (1985b). SIMSCRIPT system - declaration. LISP.6 (computer program). Cambridge, MA: Symbolics, Inc.

Cassels, R. A. (1985c). SIMSCRIPT - definitions. LISP.6 (computer program). Cambridge, MA: Symbolics, Inc.

Cassels, R. A. (1985d). SIMSCRIPT system. LISP.8 (computer program). Cambridge, MA: Symbolics, Inc.

Chao, G. (1983). High-performance computer graphics for systems design. In T. L. Kunii (Ed), Computer graphics: Theory and applications (pp 194-202). New York, NY: Springer-Verlag.

Charlesworth, A. E. & Gustafson, J. L. (1986). Introducing replicated VLSI to supercomputing: The FPS-164/MAX scientific computer. Institute of Electrical and Electronics Engineers: Computer, 19(3), 10-23.

Clocksin, W. F. & Mellish, C. S. (1984). Programming in PROLOG (2d Ed.). New York, NY: Springer-Verlag.

Cohen, P. R. & Feigenbaum E. A. (1982). The handbook of artificial intelligence (Vol. 3). Los Altos, CA: William Kaufmann.

Cooper, D. W., Kiselewich, S. J., & Weeks, R. V. (1983). The intelligent machine model (draft report). Santa Barbara, CA: General Research Corporation.

Cooper, D. W. & Pennington, R. H. (1983a). An automated decision module (ADM) for the McClintic theater model: Vol. 2, Automated decision module architecture (GRC Report No. CR-1-1087). Santa Barbara, CA: General Research Corporation.

Cooper, D. W. & Pennington R. H. (1983b). An automated decision module (ADM) for the McClintic theater model: Vol. 1, Executive summary. (GRC Report No. CR-1-1087). Santa Barbara, CA: General Research Corporation.

Cooper, D. W. & Pennington, R. H. (1983c). An automated decision module (ADM) for the McClintic theater model: Vol. 5, Knowledge base. (GRC Report No. CR-1-1087). Santa Barbara, CA: General Research Corporation.

Cooper, G. E. (1986, July 24). The Cooper Cube: A 3-D "hardware space" for combat analysis. Briefing to the Deputy Under Secretary of the Army (Operations Research), Washington, DC.

Dalton, G. W. & Thompson, P. H. (1986). Helping engineers help themselves. Institute of Electrical and Electronic Engineers: Spectrum, 23(12), 43-47.

Davis, P. K. (1984). RAND's experience in applying artificial intelligence techniques to strategic-level military-political wargaming. Paper presented at the summer computer simulation conference of the Society for Computer Simulation, Boston, MA. (DTIC No. AD-A147-272)

Dempster, A. P. (1968). A generalization of Bayesian inference. Journal of the Royal Statistical Society, Series B, 30, 205-247.

Denning, P. J. (1986). Toward a science of expert systems. Institute of Electronical and Electronic Engineering: Expert 1(2), 80-83.

Dockery, J. T. (1984). Structure of command and control (C^2) analysis. In R. K. Huber (Ed.), Systems analysis and modeling in defense, development, trends, and issues (pp 201-224). New York, NY: Plenum Press.

Dockery, J. T. & Van den Driessche, J. (1984). Use of artificial intelligence and psychology in the analysis of command and control, SHAPE Technical Memorandum STC-TM-749, Supreme Headquarter Allied Powers Europe, Hague, Netherlands: SHAPE Technical Centre.

English, H. B. & English, A. C. (1958). A comprehensive dictionary of pyschological and psychonanalytical terms. New York, NY: David McKay Company, Inc.

Epstein, J. A. (May 1986). Intelligently speaking, Digital Review.

Fain, J. D., Gorlin, F., Hayes-Roth, S. J., Rosenchein, H., Sowizral, H. & Waterman, D. (1985). The ROSIE language reference manual. (Rand Technical Report N-1647-ARPA). Santa Monica, CA: The Rand Corporation.

Faught, W. S. (1986). Applications of AI in engineering. Institute of Electrical and Electronic Engineering: Computer, 19(7), 17-27.

Forgy, C. L. (1981). OPS 5 user's manual (Carnegie-Mellon University Report CMU-CS-81-135). Pittsburgh, PA: Department of Computer Science, Carnegie-Mellon University.

Friedland, P. & Kedes, L. (1985). Discovering the secrets of DNA, Institute of Electrical and Electronic Engineering: Computer, 18(11), 49-69.

Galamos, J. A., Abelson, R. P., & Black, J. B. (Eds.) (1986). Knowledge structures. New Jersey: Lawrence Erlbaum.

Gabriel, R. P. & Masinter, L. (1982). Performance of LISP system. Proceeding of the 1982 Association of Computing Machinery, Symposium on LISP and Functional Programing (pp 123-142), Pittsburgh, PA.

Gabriel, R. P. (1985). Performance and evaluation of LISP systems (Stanford LISP performance study: Final report). Cambridge, MA: MIT Press.

Garzia, R. F., Garzia M. R., and Zeigler, B. P. (1986). Discrete-event simulation, Institute of Electrical and Electronic Engineers: Spectrum, 23(12), 32-36.

Genesereth, M. R. & Ginsberg, M. L. (1985). Logic programming, Communications of the Association of Computing Machinery, 28(9), 933-941.

Gunsch, G. H. & Herbert, B. S. (1983). A proposed military planning task simulator using the ROSS language (Report No. AFIT-GE-830-24). Wright-Patterson Air Force Base, OH, Air Force Institute of Technology. (DTIC No. 84-02-17-051)

Hadden, W. J. & Hadden, S. G. (1985). Rulemaster: Expert system for environmental regulation. In K. N. Karna (Ed.), Proceedings of the Expert Systems in Government Symposium (pp 558-566). McLean, VA: IEEE Computer Society.

Harmon, P. & King, D. (1985). Expert system: Artificial intelligence in business. New York, NY: John Wiley & Sons.

Hass, E. & Walleck, S. (1986). Ups and downs of operating south of the border. Wall Street Journal, 208(117), 28.

Hayes-Roth, F. (1981). AI for systems management (Rand Paper P 6573). Santa Monica, CA: The Rand Corporation.

Hayes-Roth, F (1981). Artificial intelligence and expert systems, Proceedings of the 1st US Army Conference on Knowledge-Based Systems for C3I. Ft Leavenworth, KS: US Army Model Management Office.

Hayes-Roth, F. (1985). Rule-based systems. Communications of the Association of Computing Machinery, 28(9), 921-932.

Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. (1983). An overview of expert systems. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), Building expert systems (pp 3-30). Reading, MA: Addison-Wesley Publishing Company.

Hendricks, D. E., Kolduff, P. W., Brouks, P., Marshak, R., & Dovle, B. (1983). Human engineering guidelines for management information systems. Aberdeen Proving Grounds, MD: Management Information Systems Directorate, Human Engineering Laboratory.

Hillis, D. W. (1985). The connection machine. Cambridge, MA: MIT Press.

Hughes, W. P. Jr. (Ed.) (1984). Military modeling. Alexandria, VA: Military Operations Research Society.

Hurd, D. A. (1985). U.S. Army Concepts Analysis Agency computer systems: Vital statistics. Unpublished manuscript.

Jackson, P. (1986). Introduction to expert systems. Reading, MA: Addison-Wesley Publishing Company.

Jakobson, G., Lafond, C., Nyberg, E., & Piatesky-Shapiro, G. (1986). An intelligent database assistant. Institute of Electrical and Electronic Engineering: Expert, 1(2), 65-79.

Johnson, G. D. (1984). SIMSCRIPT 11.5: User's Manual, Sperry UNIVAC 1100 Series Computer Systems, SIMSCRIPT Release 7.3. Los Angeles, CA: C.A.C.A.

Kahane, H. (1978). Logic and philosophy (3d Ed.). Belmont, CA: Wadsworth Publishing Company.

Kaisler, S. H. (1986). Artificial intelligence and parallel processors. Paper presented to Institute of Electrical and Electronic Engineers, Computer Society, Artificial Intelligence Subchapter, Falls Church, VA.

Kay, A. (1984). Computer software. In Scientific American computer software (pp 3-9). New York, NY: W. H. Freeman Company.

Klahr, P., Ellis, J. W. Jr., Giaria, W., Narain, S., Cesar, E. M., & Turner, S. R. (1984). TWIRL: Tactical warfare in the ROSS language (Rand Report No. R-3158-AF). Santa Monica, CA: The Rand Corporation.

Klahr, P., McArthur, D., Narain, S. & Best, E. (1982). SWIRL: Simulating Warfare in the ROSS Language (Rand Report No. N-1885-AF). Santa Monica, CA: The Rand Corporation.

Klahr, P., & Waterman, D. A. (1986). Artificial intelligence: A Rand perspective. Artificial Intelligence Magazine, 7(2), 54-64.

Kline, P. J., & Dolins, S. B. (1985). Choosing architectures for expert systems (Technical Report RADC-TR-85-192). NY: Rome Air Development Center.

Kobler, V. P. (1985). Expert system planning for the strategic defense initiative. In US Army Research Office Proceedings of the Workshop on Military Applications of Expert Systems (pp 67-70). Crystal City, VA: US Army Research Office.

Kuipers, B. (1986). Causal models and qualitative simulation (Conference Tutorial Program). Fifth National Conference on Artificial Intelligence, Philadelphia, PA.

Kuipers, B. (1986). Qualitative simulation. Artificial Intelligence, 29(3), 289-338.

LaCasse, M. (1984). The ABEL programming language: A primer (Rand Report WD-2334-NA). Santa Monica, CA: The Rand Corporation.

Larson, J. A. (1986). A visual approach to browsing in a database environment, Institute of Electrical and Electronic Engineering: Computer, 19(6), 62-71.

Lesgold, A. M. (1983). Acquiring expertise (Technical Report No. PDS-5). Pittsburgh, PA: Learning Research and Development Center, University of Pittsburgh.

Levy, S. (1984). Hackers: Heros of the computer revolution. New York, NY: Anchor Press.

LISP Machine, Inc. (1984). The LMI lambda: Technical summary. Los Angeles, CA: LMI, Inc.

Matthews, G., Manuel, G., & Krueger, S. (1986). Matching hardware to LISP yield peak performance. Computer Design, (pp 95-98).

Matthews, L., Manuel, G., & Krueger, S. (1985). AI hardware architectures: Current and anticipated. Proceedings of the Seminar on Artificial Intelligence Application to the Battlefield, pp 1.2-1-1.2-8, Ft Monmouth, NJ.

Mayaram, K., & Pederson, D. O. (1985). Speed performance of a LISP-based circuit simulation program, BLAS LISP, on the VAX 11/780, the HP9836 desktop computer, and the Symbolics 3600 LISP Machine (unpublished study), Electronic Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkley.

McArthur, D., & Klahr, P. (1982). The ROSS language manual (Rand Report N-1854-AF). Santa Monica, CA: The Rand Corporation.

McCarthy, J. (1978). History of LISP. Association of Computing Machinery, SIGPLAN Notices, 13, 217-223.

McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., & Levin, M. I. (1985). LISP 1.5 programmer's manual (2d Ed.). Cambridge, MA: MIT Press.

McClelland, J. L., & Rumelhart, D. E. (1986). Parallel distributed processing: Explorations in the microstructures of cognition: Vol. 2. Psychological and biological models. Cambridge, MA: MIT Press.

McCune, B. P., & Dean, J. S. (1985). Trends and advanced software tools, Air Force Office of Scientific Research (Contract F49620-81-C-0067). Mountain View, CA: Advanced Information and Decision Systems, Inc.

Mead, C., & Conway, L. (1980). Introduction to VLSI systems. Reading, MA: Addison-Wesley.

Metzger, J. J. (1985). Command and control in the Force Evaluation Model. Paper presented at the 53rd meeting of the Military Operations Research Society.

Meyers, W. (1986). Introduction to expert systems. Institute of Electrical and Electronic Engineers: Expert 1(1), 100-109.

Milne, R. (1985). Information management expert systems. Unpublished manuscript, Headquarters, Department of the Army Artificial Intelligence Center; HQDA (DAIM-DO), Washington, DC.

Minsky, M. (1975a). A framework for representing knowledge. In R. J. Brachman & H. J. Levesque (Eds.), Readings in knowledge representation (pp 245-262). Los Altos, CA: Morgan Kaufmann.

Minsky, M. (1975b). A framework for representing knowledge. In P. Winston (Ed.), The psychology of computer vision (pp 211-277). New York, NY: McGraw-Hill.

Moglewer, S. (1984). The role of military science and operations research in defense planning and the operational art. Military Operations Research Phalanx, 17(4), 20.

Moon, D. A. (1987). Symbolics architecture. Institute of Electrical and Electronic Engineers: Computer, 20(1), 43-52.

Moto-Oka, T. (Ed.) (1981). Fifth generation computer systems. New York, NY: North-Holland Publishing Company.

Newell, A. & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, N. J. (1980). Principles of artificial intelligence. Palo Alto, CA: Tioqa Publishing Company.

Nugent, R. O. & Wong, R. W. (1984). The Battlefield Environment Model (BEM): Army-level threat version (MITRE Working Paper 84W-00483). McLean, VA: C31 Division, MITRE Corporation.

Pavelle, R., Rothstein, M., & Fitch, J. (1981). Computer algebra. Scientific American, 245(6), 136-152.

Perry, T. S. & Wallick, P. (1985). Inside the PARC: The information architects. Institute of Electrical and Electronics Engineers: Spectrum, 22(10), 62-75.

Pylyshyn, Z. W., Bledsoe, W. W., Feigenbaum, E. A., Newell, A., Nilsson, N., Reddy, D. R., Resenfeld, A., Winograd, T., & Winston, P. (1980). Artificial intelligence. In B. W. Arden (Ed.), What can be automated: The computer science and engineering research study. Cambridge, MA: MIT Press.

Quattromani, A. F. (1982). Catalog of wargaming and military simulation models (SAGAM Report 120-82). Washington, DC: Studies, Analysis and Gaming Agency, Organization of the Joint Chiefs of Staff. (DTIC No. AD A11595)

Reiser, B. J., Anderson, J. R., & Farrell, R. G. (1985). Dynamic student modelling in an intelligent tutor for LISP programming. Proceeding of the Ninth International Joint Conference on Artificial Intelligence (pp 8-13). Los Angeles, CA: Morgan Kaufmann Publishers.

Richardson, J. J. (1983). Artificial intelligence: An analysis of potential applications to training, performance measurement and job performance aiding (AFHRL-TP-83-28). Brooks Air Force Base, TX: Air Force Human Resources Laboratory, Air Force Systems Command.

Riese, C. E. & Zubrick, S. M. (1985). Rulemaster: An expert system to aid in severe thunderstorm forecasting (Radian Technical Report ZU-RS-00025). Austin, TX: Radian Corporation.

Roussopoulos, N. (1986a). Engineering information systems. In C. Zaniolo (Ed.), Proceedings of Association for Computing Machinery Special Interest Group Management of Data (pp 1-3). Washington, DC: Association for Computing Machinery.

Roussopoulos, N. (1986b). Future directions in database systems-architectures for information engineering. Institute of Electrical and Electronic Engineers: Computer, 19(2), 7-8.

Rumelhart, D. E., & McClelland, J. L. (1986). Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1. Foundations. Cambridge, MA: MIT Press.

Schneck, P. B., Austin, D., Squires, S. L., Lehmann, J., Mizell, D., & Wallgren, K. (1985). Parallel Processor Program in the Federal Government, Institute of Electrical and Electronic Engineers: Computer, 18(6), 3-56.

Shafer, G. (1976). A mathematical theory of evidence. Princeton, NJ: Princeton University Press.

Shafer, G. (1986). Probability judgment in artificial intelligence. In L. W. Kanal and J. F. Lemmer (Eds.), Uncertainty in artificial intelligence, (pp 127-135). New York: Elsevier Science Publishers.

Shapiro, N. Z., Hall, H. E., Anderson, R. H., & LaCasse, M. (1985). The RAND-ABEL Programming Language (Rand Report R-3274-NA). Santa Monica, CA: The Rand Corporation.

Sheil, B. A. & Masinter, L. M. (Eds.) (1983). Papers on INTERLISP-D (Corporate Accession P83-00029). Palo Alto, CA: XEROX Palo Alto Research Center.

Simon, H. A. & Newell, A. (1958). Heuristic problem solving: The next advance in operations research. Operations Research, 6, 1-10.

Sippl, C. J. (1985). Computer Dictionary (4th Ed.). Indianapolis, IN: Howard W. Sams & Co., Inc.

Slagle, J. R., & Hamburger, H. (1985). An expert system for a resource allocation problem. Communications of the Association of Computing Machinery, 28(9), 994-1004.

Steele, G. L., Jr. (1984). Common LISP: The language. Burlington, MA: Digital Press.

Stefik, M., Bobrow, D. G., Mittal, S., & Conway, L. (1983). Knowledge programming in LOOPS: Report on an experimental course. AI Magazine, 4(3), 3-13.

Stefik, M., & Bobrow, D. G. (1986). Object-oriented programming: Themes and variations. AI Magazine, 6(4), 40-62.

Symbolics, Incorporated (1984). Symbolics 3600: Technical summary. Cambridge, MA: Symbolics, Inc.

Symbolics, Incorporated (1985). Bibliography of papers referencing MACSYMA. Cambridge, MA: Symbolics, Inc.

Symbolics, Incorporated (1985). Reference guide to Symbolics-LISP. Cambridge, MA: Symbolics, Inc.

Teitelman, W. & Masinter, L. (1983). The INTERLISP programming environment. In B. A. Sheil & L. M. Masinter (Eds.), Papers on INTERLISP-D (pp 15-28) (Corporate Accession P-83-00029). Palo Alto, CA: XEROX Palo Alto Research Center.

Teitelman, W. (1969). Toward a programming laboratory. Proceedings of the First International Joint Conference on Artificial Intelligence (pp 1-8). Washington, DC.

Texas Instruments Incorporated (1984). Explorer: Technical summary. Austin, TX: Texas Instruments, Inc.

Tiede, R. V. (1978). On the analysis of ground combat. Manhattan, KS: Military Affairs/Aerospace Publishing, Kansas State University.

Touretzky, D. S. (1986). The mathematics of inheritance systems. Los Altos, CA: Morgan Kaufmann Publishers.

US Army (1986). Minutes of the Army Models Committee (AMC) Executive Group Meeting. Unpublished manuscript.

US Army (1983), Office of the Deputy Under Secretary of the Army (Operations Research). Catalog of war games, training games, and combat simulations. Quadripartite Working Group on Army Operations Research, Pentagon, Washington, DC. (DTIC No. 84-03-13-187)

US Army (1983), AR 5-11. Army model improvement program. Washington, DC: Headquarters Department of the Army.

US Army (1985), Soldier Support Institute. An introduction to artificial intelligence: A self-study text. Reference Book RB 18-155, Computer Science School, United States Army Soldier Support Institute, Ft Benjamin Harrison, IN.

VanCott, H. P. & Kinkade, R. G. (Eds.) (1972). Human engineering guide (revised edition). Washington, DC: American Institutes for Research.

Vandiver, E. B. III (1985). FY 1988-1992 program analysis resource review (PAAR): Director's statement. Bethesda, MD: United States Army Concepts Analysis Agency.

Vandiver, E. B. III (1986). The CAA study process: Desired attributes. Bethesda, MD: United States Army Concepts Analysis Agency.

Wah, B. W. (1987). New computers for artificial intelligence processing. Institute of Electrical and Electronic Engineers: Computer, 20(1), 10-15.

Waterman, D. A. (1986). A guide to expert systems. Reading, MA: Addison-Wesley Publishing Company.

Waterman, D. A. & Hayes-Roth, F. (1982). An investigation of tools for building expert systems (Rand Report No. R-2818-NSF). Santa Monica, CA: Rand Corporation.

Weld, D. S. (1986). The use of aggregation in causal simulation, Artificial Intelligence, 30,(1), 1-34.

Winston, P. H. (October 1985). Personal communication between Patrick Henry Winston and the Vice Chief of Staff, United States Army.

Winston, P. H. & Brown, R. H. (Eds.) (1979). Artificial intelligence: An MIT perspective: Vol. 2, Understanding vision, manipulation, computer design, symbol manipulation (pp 337-342). Cambridge, MA: MIT Press.

Yokoi, T., Goto, S., Hasyashi, H., Kunifuji, S., Kurokawa, T., Motoyoshi, F., Nakashima, H., Nitta, K, Sato, T., Tomokatsu, S., Ueda, K., Umemura, M., &, Umeyama, S. (1982). Logic programming and a dedicated high-performance personal computer. In T. Moto-Oka (Ed.), Fifth generation computer systems (pp 159-164). New York: North-Holland Publishing Co.

Zadeh, L. A. (1981). Possibility theory and soft data analysis. In L. Cobb, and R. M. Thrall (Eds.), Mathematical frontiers of the social and policy sciences (pp 69-129). Boulder, CO: Westview Press.

Zadeh, L. A. (1985). Syllogistic reasoning as a basis for combination of evidence in expert systems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence (pp 417-419). Los Altos, CA: Morgan Kaufmann Publishers.

Zadeh, L. A. (1986). Is probability theory sufficient for dealing with uncertainty in AI: A negative view. In L. W. Kanal & J. F. Lemmer (Eds.), Uncertainty in artificial intelligence (pp 103-116). New York: Elseiver Science Publishers.

Zymelmen, A. S. (1986). An AI technology insertion experiment with analyst (MITRE Working Paper WP-85W 00641). Washington, DC: Washington C3I Operations, C3I Division, The MITRE Corporation.

APPENDIX D

ARTIFICIAL INTELLIGENCE HARDWARE ANALYSIS

D-1. An analysis of artificial intelligence (AI) hardware was made during fiscal year 1986. Only hardware that is currently available for purchase and is configured for list operations was analyzed. Nine hardware benchmarks were used in the analysis. The rationale for the criteria was to compare high-speed list processing which is enabled by hardware designed for list operations. Other features were described in the technical notes. Software and tools for AI applications were also listed.

D-2. FUJITSU ALPHA LIST/PROLOG MACHINE

a. Hardware specifications

(1) Manufacturer: Fujitsu Electronics Company, Ltd.

(2) Machine: Fujitsu "ALPHA" LISP Machine

(3) Model: Alpha

(4) Micro instruction:

(a) Instruction length:	48 bits
(b) Instruction cycle:	160 nanoseconds
(c) Control storage size	16K words

(5) Main memory:

(a) Virtual memory	16 Megabytes
(b) Real memory	8 Megabytes
(c) Page size	4 Kilobytes

(6) Stack:

(a) Logical stack	64K words
(b) Hardware stack	8K words
(c) Swapping block size	2K words

(7) Other hardware features:

(a) Machine instruction prefetch circuit

(b) Hardware multiplier and divider

b. Software Specifications:

(1) Languages

(a) UTILISP (University of Tokyo LISP)

(b) PROLOG

(2) Tools: Unknown

c. **Note:** The Alpha has no I/O devices. A host computer software must control the I/O devices according to Alpha request. The Alpha is designed as a backend processor for a large computer in a time-sharing system.

D-3. LMI LAMBDA LISP MACHINE

a. Hardware Specifications

(1) Manufacturer: LISP Machine Inc. (LMI)

(2) Machine: LAMBDA LISP Machine

(3) Models: LAMBDA, LAMBDA-Plus

(a) LAMBDA 32-bit processor, 515 Megabytes disk

(b) LAMBDA 32-bit processor, 470 Megabytes disk

(c) LAMBDA/E 32-bit processor, 140 Megabytes, 6 configurations possible

(4) Microinstruction:

(a) Instruction length	32 bits (two 16-bit macroinstructions pipelined)
(b) Instruction cycle	100-200 nanoseconds
(c) Control storage size	1,024 words (cache memory)

(5) Main memory:

(a) Virtual memory	146 Megabytes
(b) Real memory	4-32 Megabytes
(c) Page size	1,024 Bytes

(6) Stack Memory:

(a) Logical stack	Up to 127 Megabytes dynamically expandable

(b) Hardware stack	2K
(c) Swapping block size	N/A functional 128, 256, 512, 1,024 words/page

(7) Other hardware features:

(a) 64 Kilobytes by 64-bit virtual control store (programable)

(b) LISP microcompiler

(c) NUBUS (multibus: 32-bit device-independent bus)

(d) ETHERNET (10 Megabytes/sec local area network) (a star net)

(e) Tape (½-inch, 8)

(f) 474 Megabytes Winchester disk drive

(g) 515 Megabytes Winchester disk drive

(h) 300 Megabytes removable

(i) UNIX 68010 processor (FORTRAN)

(j) High-resolution color

b. **Software:**

(1) Languages

(a) ZetaLISP-Plus

(b) LISP

(c) Common LISP (DOD standard: allows interchange of programs)

(d) PROLOG

(e) UNIX (with C compiler, FORTRAN 77, Pascal)

(f) MACSYMA

(g) PICON

(h) RPME (UNIX)

(i) AI Base

(j) Flavors

(k) ObjectLISP

(2) Tools

(a) FLAVORS (a way of constructing abstract objects that bundle up procedural and declarative information and is similar to XEROX's SMALLTALK OBJECT LISP).

(b) TRACE APROPOS, Memory Region 3, and single stepping are available for software debugging.

(Note: ART, KEE, SRL, etc., third party software.)

c. Notes:

(1) Two RS-232 ports in all models.

(2) NUBUS 32-bit synchronous bus with 37.5 Megabytes/sec transfer rate.

(3) 4-gigabyte logical address space.

(4) 8088-base multitasking monitor serves as system diagnostic unit for a smart diagnostic front end.

(5) LAMBDA 3 x 3 (3 work stations on one machine running LISP/PROLOG concurrently).

(6) LAMBDA has a byte rotator, a 32-bit barrel shifter that allows a string of any number of bits from one word to be inserted into any position in another word. LISP uses this feature primarily to extract the data type from a word and use it in dispatch instructions. It is also used in examining headers of packed data structures, for instance, array headers, and compiled function descriptions.

(7) For debugging purposes, the LISP processor can be halted without loss of information. At the time of the halt, the machine state is saved. The timing microinstruction clock is stopped so that all latched information is preserved. Spy logic circuitry can then be used for unclocked transactions, finding information about the state of the halted machine to locate the problem. The machine can also be single-stepped, with the user generating clocks in order to track down a possible timing problem.

D-4. SYMBOLIC 3600 SERIES

a. Hardware Specifications

(1) Manufacturer: Symbolics, Inc.

(2) Machine: Symbolics 3600 Series

(3) Models:

(a) 3620-190B, single drive, 2-24 Megabytes main memory

(b) 3640-1613, single drive, 8 Megabytes main memory

(c) 3640-2613, dual drive, 8 Megabytes main memory

(d) 345-3645-t, single drive, 2-16 Megabytes main memory (TEMPEST)

(e) 3650-368BASE, dual drive, 2-64 Megabytes main memory

(f) 3670-1113, 8-30 Megabytes main memory

(g) 3670-1313, 8-30 Megabytes main memory

(h) 3675-515 Base, 8-120 Megabytes main memory

(i) 3610-AE, 4 Megabytes main memory (Delivery System only)

(4) Microinstruction:	
(a) Instruction length	112 bits
(b) Instruction cycle	180-250 nanoseconds
(c) Control storage size	1K word stack buffers (see note)
(5) Main memory:	
(a) Virtual memory	1 Gigabytes virtual memory address space
(b) Real memory	2-60 Megabytes
(c) Page size	256 36-bit words
(6) Stack Memory:	
(a) Logical stack	2K words
(b) Hardware stack	N/A (varies)
(c) Swapping block size	N/A (grows as more swapping space is used)

(7) Other machine features:

(a) Stack buffers

(b) 60 Megabytes - 4 Gigabytes disk

b. Software:

(1) Languages

(a) Zeta LISP

(b) Common LISP

(c) InterLISP-D (compatibility package)

(d) FORTRAN 77

(e) Pascal

(f) MACSYMA

(g) PROLOG

(h) Ada

(i) C

(2) Tools:

(a) FLAVORS: Language features to support object-oriented programing style.

(b) ART, KEE, S.1, OPS-5e, Knowledge Craft, Hyper Calc

(c) Interactive debugging tools:

1. Trace facility: finds callers of a particular set of functions.

2. Apropos: finds all of the symbols whose name contains a given substring (can use part of a name).

3. Memory regions: marking.

4. Single stepping.

c. Notes

(1) The 3600 is built to execute large programs that need high-speed symbolic and numeric computation. Benchmarks with numeric simulations (Mayaram & Pederson, 1985) indicate equal performance with general purpose hardware.

(2) Hardware-assisted garbage collection uses garbage collection tables for recovery of memory space.

(3) Because the 3600 hardware has an organization of memory that is stack-oriented, it has no general purpose register at the macroinstruction level. Many instructions fetch their operands directly from the stack.

(4) Hardware does tag checking in parallel. This speeds up numeric calculations.

(5) Extensive color graphics system. Full window system functionality support.

D-5. TEKTRONIX MICRO LISP MACHINE

a. Hardware Specifications

(1) Manufacturer: Tektronix, Inc.

(2) Machine: 4404 Artificial Intelligence System

(3) Microinstruction:

(a) Instruction length	32 bits
(b) Instruction cycle	Unknown nanoseconds
(c) Control storage size	Not available

(4) Main memory:

(a) Virtual memory	32 Kilobytes
(b) Real memory	1-2 Megabytes
(c) Page size	Not available

(5) Stack:

(a) Logical stack	Not available
(b) Hardware stack	Not available
(c) Swapping block size	Not available

(6) Other features:

(a) RS-232C interface

(b) 40 Megabytes 5¼-inch hard disk

(c) 320 Kilobytes floppy disk

(d) Ethernet interface

(e) M68010

b. Software:

(1) Smalltalk-80

(2) PROLOG

D-6. T. I. EXPLORER

a. Hardware Specifications

(1) Manufacturer: Texas Instruments, Inc.

(2) Machine: Explorer LISP Machine

(3) Model: Explorer, VLSI Chip version to be released 1987

(4) Microinstruction:

(a) Instruction length	56 bits
(b) Instruction cycle	300 nanoseconds; 137 nanoseconds
(c) Control storage size	16K words (56 bit x 16K)

(5) Main memory:

(a) Virtual memory	128 Megabytes
(b) Real memory	2-16 Megabytes
(c) Page size	1 Kilobytes (adjustable)

(6) Stack Memory:

(a) Logical stack	128 Megabytes
(b) Hardware stack	4 Kilobytes x (32 bit x 1K)
(c) Swapping block size	1 Kilobytes

b. Software Specifications:

(1) Languages

(a) Zeta LISP

(b) Common LISP

(c) PROŁOG

(2) Tools:

(a) FLAVORS

(b) Standard MIT packages for software development and debugging such as TRACE, APROPOS, single stepping, and memory regions is also available:

(c) MACSYMA

(d) KEE, SRL, ART

c. Notes:

(1) In August 1986, Texas Instruments delivered a VLSI LISP computer to DARPA. The compact LISP machine consisted of four modules: CPU, MAP/CACHE, 4 Megabytes in memory, and multibus I/O.

(2) The compact LISP machine will be 8 to 10 times faster than the present Explorer LISP machine and will contain 144˙256K dynamic RAM VLSI chips (4 Megabytes). System developers can use VLSI custom chip in building their own LISP machine.

D-7. XEROX D SERIES LISP MACHINES

a. Hardware Specifications

(1) Manufacturer: XEROX

(2) Machines:

(a) 1100 Series

(b) 6085 Series

(3) Models:

(a) Dandelion

(b) Dophin

(c) Dorado

(4) Microinstruction:

(a) Instruction length	48 bits (12 bits is for intermediate next instruction address)
(b) Instruction cycle	137 nanoseconds
(c) Control storage size	4-12 Kilobytes

(5) Main memory:

(a) Virtual memory	32 Megabytes
(b) Real memory	1-3.5 Megabytes
(c) Page size	512 bytes (local disk)

(6) Stack Memory:

(a) Logical stack	N/A
(b) Hardware stack	16 registers
(c) Swapping block size	N/A (does not swap)

(7) Other features:

(a) 80 Megabytes rigid disk drive

(b) RS-232C serial port

(c) 10 MHz Ethernet

(d) 42 Megabytes rigid disk

(e) 1.2 Megabytes double sided, double density floppy disk drive

(f) 4K control store extension

b. Software Specifications:

(1) Languages

(a) InterLISP-D

(b) Common LISP

(2) Tools/Toolkits:

(a) Loops, a knowledge programing system

(b) Notecards, an InterLISP-D environment for authoring information structuring)

(c) Runs KEE, ART, SRL.

c. **Notes:**

(1) Transactional (interactive) garbage collection

(2) Has RS-232C serial port (9600 baud).

(3) Extended programing aids are available:

(4) The 1108 with 3.5 Megabytes of main memory is designed to avoid swapping.

(5) Avoids swapping in the design of its main memory.

(6) Hardware called the spaghetti stack allocates memory contiguous to main memory using 16 bit stack pointers to 64K byte blocks. Lots of stacks may be allocated in main memory as part of the spaghetti. A stack's 128-byte length cannot be changed. Think of the stacks as a tree of linked objects called frame extension.

(7) A TEMPEST version of the XEROX is available:

D-8. DARPA BUTTERFLY PARALLEL PROCESSOR

a. Hardware Specifications

(1) Manufacture: Bolt, Beranek, and Newman, Inc.

(2) Machine: Butterfly Parallel Processor

(3) Models	3 Node
	4 Node
	16 Node
	32 Node
	64 Node
	128 Node
	256 Node

(4) Microinstruction

(a) Instruction Length:	32 bits

(b)	Instruction cycle:	500K Instruction/sec/node
(c)	Control Storage Size:	16-bit (8-bit node specific and 8-bit sequence number)

(5) Main Memory

(a)	Virtual Memory:	N/A
(b)	Real Memory:	256 - 1024 Megabytes
(c)	Page Size:	N/A

(6) Stack Memory

(a)	Logical Stack:	Local to each node
(b)	Hardware Stack:	Local to each node
(c)	Swapping Block Size:	N/A

(7) Other Machine Features

(a) The Butterfly multiprocessor is a tightly coupled shared memory machine where each of up to 256 processor nodes (with up to 4 Megabytes/node) can work concurrently on a single algorithm. Each processor node adds processing power to the total architecture. For example, a near linear speedup was demonstrated for matrix multiplication using 400 x 400 matrices for up to 128 processor nodes. A single physical address (up to 16 Megabytes) manages the memory for all processor nodes.

(b) All memory is local to the processor node. Each node can address the memory of any other processor node.

(c) Each processor node executes its own sequence of instructions on possibly different data without having to call another node. The Butterfly is therefore a multiple instruction stream, multiple data stream machine (MIMD).

(d) An internal network of switches is used to communicate among processor nodes. The switch design is congruent to the format of a fast fourier transform. The switch design of the Butterfly allows fast access to all the memory of the machine.

(e) The physical switching configuration of the Butterfly is a 4 x 4 (4 input/4 output) crossbar switch node. The switch has a 32-megabit per second (32 Mbps) bandwidth.

(f) The butterfly was designed as a backend processor. A VAX, Sun, or Symbolic/LMI LISP machine is required for a host.

c. Notes

(1) The CHRYSALIS operating system is designed to distribute tasks among the processors during program execution. This operating system is based on UNIX with all objects using a 32 bit address. The operating system uses an interactive garbage collector. The operating system also supports communication and synchronization between the processor nodes of the Butterfly.

(2) **The Uniform System** of storage management allows the distribution of tasks to processors based on efficiency rather than physical location of the processor nodes. This feature assists in loading balancing among the processors and allows the total system of processor nodes to work together in uniform efficiency.

(3) An extensive set of debugging tools for the Butterfly is currently under development.

D-9. THE CONNECTION MACHINE

a. Hardware Specifications

(1) Manufacturer:	Thinking Machines Corporation
(2) Machine:	The Connection Machine
(3) Model:	CM-1
(4) Microinstruction	
(a) Instruction Length:	32 bits
(b) Instruction Cycle:	1,000 MIPS
(c) Control Storage Size:	4 x 4 Kilobytes per cell
(5) Main Memory	
(a) Virtual Memory:	N/A
(b) Real Memory:	250 Megabytes
(c) Page Size:	N/A
(6) Stack Memory	
(a) Logical Stack:	N/A
(b) Hardware Stack:	N/A
(c) Swapping Block Size:	N/A

(7) Other Machine Features

(a) The Connection Machine (CM-1) contains 16,384 or 65,536 microprocessor cells. Each cell contains 4 Kilobytes of memory and a microprocessor.

(b) The Connection Machine architecture allows the microprocessor cells to process data. The processing elements called cells are used to represent and process data stored within them. Cells are connected by programable communications networks. Multiple cells are used to represent and process data. The network of data driven patterns of cells are called ACTIVE DATA STRUCTURES. Computation proceeds by the interaction of the cells in the data structures.

(c) Connections can be determined dynamically by external input or by the machine itself. The Connection Machine's ability to dynamically change its own connections implies that applications in self-modifying (e.g., self-learning) systems are possible.

(d) The Connection Machine can be customized by varying:

1. Number of processing/memory cells;

2. Memory per processor; and

3. Size of each processor.

(e) Connection machines can be built with several hundred Megabytes of memory and several million processors. The trade-off of the three above variables is important since Connection Machine data structures are built up by linking multiple processors.

b. Software Specifications

(1) Languages

(a) Connection Machine LISP (CmLISP)

(b) C language

(c) FORTRAN

(2) Tools: Ext Kernel (EXT) Expert System Development Tools

c. Notes

(1) Connection Machine LISP (CmLISP) is based on Common LISP. The programer uses simultaneous operations over composite data structures (i.e., sectors) to achieve parallel programs. This approach is called data-level parallelism. This methodology is different from conventional programing approaches to parallelism which use concurrent control structures.

(2) The control structure of CmLISP is similar to APL.

(3) The Connection Machine can perform roughly 1 billion cell updates per second. A simulation involving 100,000 steps can take less than 30 minutes.

(4) Traditional Operations Research techniques such as the simplex algorithm and MAX/MIN in order one time have been prototyped on the Connection Machine. The machine allows data level simulation.

(5) The internal communications mechanism is called a router. It has a total capacity of 3 billion bits/sec in passing data between processors.

D-10. INTEL IPSC HYPERCUBE

Note: After the completion of the AI Study, the Intel Corporation announced that their concurrent "personal supercomputer" had expanded its parallel programing environment to include artificial intelligence applications. Intel and Gold Hill Common LISP are currently developing a concurrent Common LISP to run on the Intel Cube Machine. Technical information on the nine hardware benchmarks used in the hardware analysis was requested from the company, but this information was not received in time for inclusion in this appendix.

D-11. MITSUBISHI PERSONAL SEQUENTIAL INFERENCE (PSI) MACHINE

a. Hardware Specifications

(1) Manufacturer: MITSUBISHI ELECTRONICS

(2) Machine: MELCOM PSI Prolog Mainframe

(3) Model: PSI

(4) Microinstruction:

(a) Instruction length:	40 bits
(b) Instruction cycle:	30K LIPS
(c) Control storage size	N/A

(5) Main Memory

(a) Virtual memory	N/A

(b) Real memory	40 bits x 16M words
(c) Page size	1024 words
(6) Stack	
(a) Logical stack	N/A
(b) Hardware stack	N/A
(c) Swapping block size	N/A

(7) Other hardware features:

(a) Standard IEEE-796 Bus is adopted for I/O interface.

(b) PSI is capable of 64 different internal data types that are directly manipulated by hardware/firmware.

(c) PSI machine has a 32 bit logical address space. The logical address space is divided into 256 independent areas which can be expanded to maximum of 16M words.

(d) The memory management system uses a heap area of four stacks. The interpretation of the kernal PROLOG requires all four stacks. This limits the machine to a maximum of 63 processes at a given time.

(e) PROLOG unification and backtracking are performed by machine hardware and firmware.

b. Software Specifications:

(1) Languages: Extended Self-contained PROLOG (ESP)

(2) Tools: Ext Kernel (EXT) Expert System Development Tools

c. **Notes:**

(1) The ESP language permits both object-oriented and predicate expressions. It can also be called from the Expert System Development Tools (EXT).

(2) The execution speed of a compiled PROLOG program is approximately 30,000 LIPS (Logical Inferences Per Second).

(3) The PSI has no operating system. Logic programing language is considered undesirable for operating systems. PROLOG-like languages allow programers to processes partial procedures. The machine derives an answer by using hardware, firmware, and software to examine the possible alternatives using fragmentary rules. This notion does not fit the deterministic procedures found in most operating systems.

(4) The local/global stacks are used for locating variable cells dynamically. The scope of each variable is localized within the clause, and its variable cell is allocated in the local/global stack areas when that clause is called from some other clause. This interpretation mechanism requires that several stack frame base addresses should be kept in working registers. They are local stack frame base, global stack frame base, objective code address, etc.

(5) There are two methods of representing structured data, one is structure sharing and the other is structure copying. Each method may be used by the programer to enhance the characteristics of the program. Sharing, for example, is not suitable for programs that frequently use large structured data and infrequently access their elements. Typical copying overhead might exceed the access overhead of shared structure elements. In structure sharing the variable cells contain two pointers. One points to the value area in the global stack and the other to the structure representations. The structured data are represented by actual values and its structure representation.

(6) Inference is defined by the system as the "mechanism that is used to execute a program written in logic programing."

(7) In backtracking the machine holds the entire execution history. This includes a record of each choice point and the binding environments of the variables. No efficient algorithm has been developed.

(8) The PROLOG unification is performed by primitive operations in the PSI machine that search for the called clause, fetch the arguments of the caller and callee, and examine the equality of the arguments. The use of tagged data allows greater speed in the unification process.

APPENDIX E

CAA MODELS

The following tabulation summarizes models of interest to the US Army Concepts Analysis Agency (CAA). The spectrum of models reviewed reflects the diverse analytic mission of the agency. The table can be interpreted with the following code:

Code	Interest Category
A	Active - available in-house and used frequently or on a regularly recurring basis:
B	Conversion/Test - undergoing conversion or testing for in-house use.
D	Development - being developed by CAA and/or contractor.
I	Interest - developed by other agencies or firms; potentially useful to CAA.
O	Outside - used by CAA at facility outside CAA; not available in-house.
S	Stand-by - models available in-house; not currently in active use.

Suffix G	computer graphics program
Suffix M	model or multimodel system
Suffix S	support routine
Suffix U	utility routine
Note (gi)	interactive graphics options(s) included
Note (go)	supplemental graphics options(s) included

(NOT USED)

TABULATION OF MODELS
OF INTEREST TO CAA

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
ALLOCATION		CAA 1983		AM	A model to allocate non-regimental personnel spaces to US Army regiments. The allocation objective is to equalize career opportunities of soldiers regardless of regimental affiliation.
ASM (MRFS)	Acquisition Strategy Model (Mid-Range Force Study)	CAA 1983	150K	AM	The Acquisition Strategy Model designs yearly incremented, deployable Army forces to transition from a selected starting year structure to a selected ending year structure, in accordance with a specific transition policy. The model considers the multitheater strategy, as it is determined for each of the transition years, while it designs the time-phased theater forces. This model employs all of the features of the force design model, but on a multiyear, rather than a single year basis.
ATCAL	Attrition Calibration	CAA 1983		D	Generates simulated combat attrition results, suitable for use in a theater level simulation over a range of weapon and force mixes, based on combat parameters derived from a small set of results from a high resolution combat model.
ATLAS	A Tactical, Logistical, and Air Simulation	RAC, for ODCSOPS 1969-70 (as part of the FOREWON system)	52K	AM (go)	A theater level model for force effectiveness, composed of four interacting simulations: ground combat, air combat, logistics, and theater control. This deterministic model plays opposing forces in ten independent sectors and measures FEBA movement, loss of firepower, and personnel attrition. Forces input to the model are described in terms of weapons scores and personnel strengths.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
CAMP	Computer-Assisted Match Program	CAA (STAG) 1972-73	68K	AS	A data processing system which interrelates FASTALS output with information from other data bases, such as Force Accounting System (FAS) and Type Unit Characteristics (TUCHA), to generate unit movement requirements in acceptable format for input to strategic mobility models.
CARMONETTE	CARMONETTE (ATHELO)	RAC CARMONETTE I 1960	64K	AM	A computerized, Monte Carlo, time-sequenced, critical event simulation of a combined arms air/ground war game. Played on a terrain representation of 60 x 100 grid squares at 100 meters resolution for an hour of combat engagement. Force representation of infantrymen or various vehicles including tanks, armored personnel carriers, air defense, and helicopters at individual-resolution in a battalion-level situation. Events pertain to surveillance, movements, communication, and weapon activities. Surveillance considers the effects of battlefield obscuration including weather, aerosol smokes, and artillery dust. Probabilities of hit and kill consider the biased dispersion of weapon systems based on moving firer/targets. Output consists of displays and detailed reports including the killer/victim scoreboard.
CEM V	Concepts Evaluation Model	GRC, CEM for ACSFOR 1971	150K	AM	A two-sided, fully automated, deterministic model capable of aggregating conventional warfare results as a series of 4-day theater level cycles. The theater cycle includes three subcycles down to a 12-hour division cycle. Force estimation and decision processes are simulated at all four command cycles. The model accepts input data in terms of battalion on the Blue side and regiment on the Red side, requires a continuous FEBA (piece-wise linear sections), and simulates combat between Blue brigades and Red divisions over 12-hour time increments using firepower potentials (FPP), modified by environmental and combat conditions, to assess casualties and FEBA movement. Other effectiveness measures reported are loss of major weapons (41 types), changes in unit status, and resource expenditures. Artillery, helicopter, and tactical air fire support are provided. Three types of terrain, equipment and ammunition resupply, maintenance and repair, personnel and unit replacement, and medical support are also simulated.
(ADDCOP)	Automated Data Display of CEM	CAA 1975	N/A	AGU	A subroutine in the CEM report writer that produces filed data in proper format for processing by a graphics display program.

*CAA has primacy.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	De cription of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
(CEMIFP)	CEM Firepower Computations	CAA 1976	35K	AU	A collection of utility routines to calculate basic weapon system firepower scores and ammunition expenditure rates for CEM inputs, to include data for unit firepower routines and manipulation of weapons data input files.
(CUT)	Combat Unit Trace	CAA 1975	30K	AU	Using unit data filed by the CEM report writer, this program produces a trace of combat unit activity for those units designated by the user. Output includes FEBA position, mission, and logistics information.
CEM VI (CEM/ATCAL)		CAA 1983	260K	D	An adaptation of CEM V which accepts killer/victim data produced by a high resolution model. Extrapolations from this data are made to determine losses during the simulation of combat between BLUE Brigades and RED Divisions. During the engagement simulation, weapon values are determined and are used to assess FEBA movement. Effectiveness measures reported and command cycles portrayed are the same as CEM V.
CHEMCAS II	Chemical Casualty Assessment Model	USA Ordnance Center & School 1975	60K	S	A s ries of programs to array small unit target forces, build chemical clouds, and assess effects of chemical volleys by user-input delivery systems.
COSAGE				AM	A high resolution (division-level) simulation model which simulates a day's combat activity to generate ammunition consumption, and equipment and personnel loss data. COSAGE is a two-sided, stochastic model written in the SIMSCRIPT II.5 programing language. The determining factors for the core requirements is the number of concurrent small unit battles which occur, the total number of units played (from platoon to division level), and the number of orders for these units. Orders are given for maneuver units; and if given at the battalion level, for example, will be executed by every maneuver platoon under that battalion. The gamer can direct a mane ver unit to do one of three things: (1) move; (2) attack; (3) defe d. Indirect fire units act and react dynamically. In addition to c mbined arms ground combat, COSAGE can simulate attack helocopters, mines, visibility restructions, weather conditions, smoke, illu ination, tactical aircraft and air defense.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Dɩ scription of model characteristics. Remarks include ongoing eɩforts, or future plans for significant improvements, but do nɩ t include minor modifications.
COST MODELS	Cost Models				A :ollection of models, programs, and utility routines used fc· costing weapons systems and forces.
(ADCOST)	Air Defense Cost Model	CAA 1976	20K	S	A ɩodel which calculates the total life cycle cost estimate for a group of weapon systems.
(COSFAM)	TOE Cost Factors Model	CAA 1973-74	28K	S	Accesses the UDS data bank and develops cost factors by type TOE unit.
(COSTR)	A Structured Cost Model	CAA 1979	16K	S	A model to calculate the costs of acquiring weapons systems and operating them in a force for a variable time.
(COSTSRCS)	Costs Standard Requirements Codes	CAA 1978	35K	S	Computes the cost of discrete or multiple SRC force units. Accesses the CAA cost data bank and displays costs for options selected.
(FASMATCH)	FASTALS Input Comparison	CAA 1979	35K	S	Lists the FASTALS SRCs which are not in the CAA cost data base.
(FORCOST)	Force Costing Model	CAA 1980	80K	S	An ASCII FORTRAN model that provides in-house implementation of the Comptroller of the Army, Force Cost Information System (FCIS). FORCOST provides the cost of existing, modernized, and new SRC units. It provides the cost of a single SRC unit or aggregates the cost of m:ltiple SRC units to provide force unit costs. The TOE Data Base may be used to define SRC units for costing. Cost reports are available in the traditional FCIS matrix or in an abbreviated matrix at the user's option.
(JAEX)	Joint Force Cost Extract	CAA, 1973	40K	S	Compares a troop list contained in the UDS with CAA's cost data bank and provides cost data in UDS format as output. Mi:matches are printed out.
(MODEXIT)	Modifications or activation of existing TOE/SRC units	CAA, 1973	70K	S	Calculates detailed cost data for a modernized force unit an: updates the Cost Data Bank.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
*FASTALS	Force Analysis Simulation of Theater Administrative and Logistic Support	RAC, for ODCSOPS 1969-70 (as part of the FOREWON system)	120K	AM	Computes time-phased logistic and administrative workloads for an active theater and rounds out ATLAS-generated (or CEM-generated) combat force with administrative and logistic units to perform the workloads. CAA has made significant modifications/changes to the model, associated postprocessors, utility routines, and model interface procedures since assuming responsibility for FOREWON in 1970-71.
FDM (MRFS)	Force Design Model (Mid-Range Force Study)	CAA 1983	70K	AM	The Force Design Model focuses on a single year and, considering the selected multitheater strategy, designs a deployable Army force in terms of its theater-level combat forces together with explicit combat support and implicit combat service support. This global, time-phased model uses threat estimates, resource projections, theater terrain factors, strategic lift characteristics, war reserve stocks factors, and an indicator of readiness.
FORCEM	Force Evaluation Model	CAA 1983		D	The Force Evaluation Model (FORCEM) is the theater level component of the hierarchy of Army analysis models, designed to be used with the Corps/Division Evaluation Model (CORDIVEM) and the Combined Army and Support Task Force Evaluation MOdel (CASTOFOREM) in a three level, linked hierarchy. The model is a two sided, deterministic representation of land and air combat and support operations for an extended (up to 180 days) theater campaign. The complete range of combat, combat support and combat service support activities from ports and airfields to the FLOT is represented. The level of resolution for combat organizations is the division; CS and CSS units are represented down to battalion level. Terrain, environment and weather factors are considered. The model is primarily a time step simulation with a variable step size (presently 12 hours minimum). Command and control are modeled in a four level automated decision process for Theater, Army/Front, Corps/CAA and Division. Units move freely under control of the decision logic and user specified unit boundaries, phase lines, etc. Individual weapon systems are considered in combat engagements where attrition is derived from results of Corps/Division component of the hierarchy. The model is programed principally n SIMSCRIPT and is now in the initial stages of validation testing.
ICSM	Improved Casualty Stratification Model	CAA 1983	12K	AM	A table driven model using factoring techniques to stratify various types of casualties, evacuees, and returns to duty into selected occupational category sets. Occupational categories may be defined by career management field or by MOS. The model is a part of a casualty estimation system that also utilizes FASTALS, the Patient Flow Model and the CAA TOE data base.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
MOBREM	Mobilization Base Requirements Model	CAA 1980	50K	D	A model/data base bringing together the data and programs to identify the work loads that will be placed on the CONUS Base during mobilization. The programs assign the workloads by size and time period to CONUS Base Installations and utilize mobilization manpower and equipment standards to calculate the support required by the workloads. The model is deterministic, using each workload as an argument in the standards to calculate the support required--one time period at a time. It assumes the calculated support for each time period is made available in the succeeding time period as an additional workload. This process is continued through six 10-day time periods (M to M 60) and seven 30-day time periods (M+90 to M-270). The initial version of the model is limited to requirements determination unconstrained by assets or facilities. A parameter file enables the user to evacuate the impacts of mobilization policy changes.
Overview/ ARLCAP	Overview Army Logistics Capability Model	CAA 1983	200K	AM	The Overview/ARLCAP (Army Logistics Capability) Model is a deterministic, time-stepped simulation model designed to estimate the flying hour capability of the Army's helicopter fleet given various inventory levels of spare parts. It is a modified and enhanced version of the Overview/Mission Degradation Model used by the Air Force to support its budget requests for repairable spares and to perform analyses of mission capability. Overview is a macro-level model which aggregates parts inventories at two echelons, wholesale (depot) and retail (AVIM/AVUM).
PRISM	Army Prisoner Management Model	CAA 1983	100K	S	A network simulation model constructed within the context of Q-GERT. The model assesses the impact of various changes in confinement policy decisions and environmental conditions in the criminal justice system on the prison populations.
Q-GERT	Queueing Graphical Review Technique		50K	AI	Developed by Prof. A Dritsker to solve complex network analysis problems.
RIM	Readiness Indicator Model	CAA 1977	70K	AM	A deterministic model for predicting unit deployment status, and sustainability of the current force. RIM depicts the redistribution of major items of equipment and personnel by occupational specialty. Numerous options for distributing assets can be analyzed by varying model parameters.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
SACM	Sustained Air Campaign Model	CAA 1983	TBD	D	An analytical model designed to determine missile expenditures and system losses of surface-to-air missile (SAM) systems on a daily basis when countering a scheduled air threat. The model uses expected values for SAM systems performance over a day of air war. The threat air schedule is normally obtained from the CEM model for the period following the initial high intensity air campaign. Output also includes aircraft kills by SAM systems plus kills from air-to-air combat (using air intercept productivity factors). The model is undergoing test and application in the 1990 requirements studies.
SITAP	Simulation for Transportation Analysis and Planning	Computer Science Corporation 1968	150K	AM	SITAP is primarily a deterministic model of an intratheater transportation system. The nodes at which cargo begins and/or terminates movement are modeled at a level of detail to include usage of equipment such as cranes and forklifts. The movement of cargo or vehicle from node to node is also modeled. The use of the model is to determine the throughput of airports, seaports, and the intratheater transportation network. Cargo movement requirements should be obtained from a deployment model. The model is supported by the Defense Communication Agency's Command and Control Technical Center.
TARMS-II	TRASANA Aircraft Reliability Maintainability Simulation-II	CAA 1983	250K	AM	The TARMS-II Model is a large scale, microlevel, stochastic model which describes the impact and interaction of reliability, maintainability, and vulnerability parameters on the availability and resource utilization of current or future Army aircraft. The model is sensitive to aircraft utilization rates, component failure rates and repair times, manpower quantities, support and test equipment resources, inventory and supply policies, scheduled maintenance policies and combat damage caused by single projectile and proximity fused threat weapon systems. The model is an extension of the ARMS-II and TARMS models, is written in SIMSCRIPT II.5 and includes data postprocessors.
TRANSMO	Transportation Model	CAA 1977 From earlier versions dating from 1973	63K	AM	An intertheater strategic mobility model in which movement requirements are identified in terms of tonnage by cargo type. Major features include the application of user specified attrition factors to air and sealift assets, regeneration of lost cargo, convoy and/or independent vessel shipping policy, origin port constraints, troop deployments, automatic generation of resupply requirements based on specified consumption factors and a desired build-up policy, and detailed reports. Projected improvements include a vehicle priority preference option, increased number of utilization rates and model documentation.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
*UDS	Unit Data System	CAA 1973	20-60K	AS	Programs, routines, and data files interrelating force and equipment data in DA/DCSOPS and/or CAA designed force structures.
(BUILD)	Build		30K	AS	Converts and modifies forces for UDS use.
(EXTRACT 1)	Extract 1		40K	AS	Extracts and displays data on selected force.
(EXTRACT 2)	Extract 2		50K	AS	Extr cts and displays data on selected force and authorized equi ment (organizational equipment list (OEL)).
(EXTRACT 3)	Extract 3		60K	AS	Extr cts and displays data on selected force, authorized equi ment and current equipment assets.
(MOD USER PROG)	Model User Program		35K	AS	Tail red extract and sort routines.
(UPDATE)	Update		30K	AS	Edits and updates UDS files.
WAFF	Wartime Fuel Factors Postprocessor	CAA 1980		AM	Develops fuel factors used in calculating fuel war reserves. Uses outp t from existing CAA high resolution and theater combat models for cenario-related loss rates of equipment in forward areas. Uses historical data for losses in rear area from WARF.
*WARF	Wartime Replacement Factor System	CAA 1973-75		AM	Produces material attrition factors by combining historical loss rates and rates derived by combat simulation. Uses existing CAA high resolution and theater combat models for scenario-related loss rates of equipment in forward areas. Uses historical data for losses in rear areas and/or from causes other than hostile fire. Includes support routines for automated interface of system components.
(CEM)				AM	See CEM V.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
(MATGEN)	Loss Rate Matrix Generator Program	CAA 1974	13K	AS	Produces an edited file of historical loss data from which data replaceable by simulation is excluded.
(SYMWAR)	System for Estimating Materiel Wartime Attrition and Replacement Requirements	RAC (GRC) 1969	20K	AM	Provides historical loss data for equipment classes by cause, combat posture, and theater zone. Contains routines modified by CAA to support WARF system.
WARRAMP	Wartime Requirements for Ammunition, Materiel and Personnel	CAA 1978		D	A methodology for estimating combat requirements for conventional ammunition, equipment replacements, and petroleum. The methodology includes: a two-level (division and theater) hierarchy of simulation models; a free-standing analytic model; preprocessors and postprocessors for productions of engagement, tactical and logistical reports and selective specialized reports to be used in production of WARFs, ammunition rates, petroleum requirements, and other data. The division model is COSAGE and the theater model is CEM, both described below. The postprocessors include ammunition, WARF and WAFF Models described elsewhere in this tabulation.
(CEM)					See CEM V
(COSAGE	Combat Sample Generator	1978-80	262K (dynamic)	AM	A high resolution (division-level) simulation model which simulates a day's combat activity to generate ammunition consumption, and equipment and personnel loss data. COSAGE is a two-sided, stochastic model written in the SIMSCRIPT II.5 programing language. The determining factors for the core requirements isthe number of concurrent small unit battles which occur, the total number of units played (from platoon to division level), and the number of orders for these units. Orders are given for maneuver units; and if given at the battalion level, for example, will be executed by every maneuver platoon under that battalion. The gamer can direct a maneuver unit to do one of three things: (1) move; (2) attack; (3) defend. Indirect fire units act and react dynamically. In addition to combined arms ground combat, COSAGE can simulate attack helicopters, mines, visibility restrictions, weather conditions, smoke, illumination, tactical aircraft and air defense.

Acronym	Name	Origin	Computer memory required	Nature of CAA interest	Description and remarks
Popular identifier	Full name of model	Agency and/or model antecedent and date of origin (if known)	K = 1000 computer words	Category code	Description of model characteristics. Remarks include ongoing efforts, or future plans for significant improvements, but do not include minor modifications.
(APP)	Ammunition Post Processor	CAA 1979-80		D	The APP uses CEM outputs of attrition and activity levels and will also use COSAGE to provide required inputs to the final report generator.
(MPP)	Materiel Post Processor	CAA 1979-80		D	The MPP will produce total theater losses of major items of equipment based on CEM and COSAGE outputs of attrition.

APPENDIX F

A TAXONOMY OF GUIDELINES FOR DEVELOPING EXPERT SYSTEM SOFTWARE IN A MILITARY ENVIRONMENT

This taxonomy was developed by Kline and Dolins (1985). It outlines 47 guidelines with supporting details for developing Military Expert System Software. This taxonomy is also applicable to general expert system software development. Many of the details of the taxonomy are developed in greater detail by Hayes-Roth, Waterman, and Lenat (1983) and Waterman (1986). Examples of the successful applications of expert system software may be found in Jackson (1986).

A Taxonomy of Guidelines for Developing Expert System Software in a Military Environment

A.0 General Description

B.0 Data

B.1 Description of data:
- population (from which samples are drawn)
- samples
- data points

B.2 Volume of data
- samples per unit time
- data points per sample

B.3 Uncertainty in population, samples, and data

B.3.1 Redundancy of data
- explicit (same data repeated)
- implicit (different data pointing to the same conclusions - reasons repeated)

B.3.2 Conflicting data

B.3.3 Ambiguity in data

B.3.4 Erroneous data

B.3.5 Missing data

B.3.6 Reproducibility
- amount of noise introduced by reproduction
- system (in data collection and processing)
- environment

B.4 Overdetermined or underdetermined problem

B.5 Single set of data vs. multiple data inputs (overtime)

B.6 Continuous vs. discrete data

B.7 Level of input (signal/sensor data, message level or symbolic inputs, English text, etc.)

A Taxonomy of Guidelines for Developing Expert System Software in a Military Environment

B.8 Capabilities for experimentation (obtaining additional information)

B.8.1 Sample more data from current set

B.8.2 More processing of original data

B.8.3 More processing of new data

B.8.4 Request additional information

B.9 Nature of environment (cooperative, passive, hostile)

C.0 Hypotheses

C.1 Requirements

C.1.1 General nature of hypothesis (use or purpose)

C.1.2 Is there a unique solution
- unique solution in nature
- unique solution exists from input data

C.1.3 Hypothesis nature
- best or satisfying answer required?
- one or many hypotheses formed?
- complete set of hypotheses generated or stop when good enough (satisfying) answer found?
- are hypotheses ranked?

C.2 Specific characteristics

C.2.1 Type of hypothesis
- descriptive
- functional
- causal (strategic or tactical)
- teleological (purposeful or goal directed)

C.2.2 Form of primitives

C.2.3 Syntax of hypothesis

C.2.4 Complexity of hypothesis
- number of primitives, other elements
- hypothesis structure

A Taxonomy of Guidelines for Developing Expert System Software in a Military Environment

C.2.5 Number of hypotheses (multiple hypotheses)

C.2.6 Size of hypothesis space

C.2.7 Precision of satisfaction criteria

C.2.8 Uncertainty
- quantitative vs. categorical

D.0 Reasoning Process

D.1 Form of abstract reasoning model (high-level description of the reasoning process)

D.1.1 Uncertainty

D.1.2 AI paradigm
- top-down vs. bottom-up (goal-directed vs. data or event driven)
- search strategy (breadth first, depth first, etc.)
- confirmation strategy
- evidence gathering
- predictions

D.2 Specific reasoning (or control) structures

D.2.1 Hypothesis formation technique
- selected (from pre-stored list)
- instantiated (from template)
- generated (from primitives)
- analogy (abstracted from instances)

D.2.2 Size of space considered
- pruning or constraints needed
- generator vs. model based

D.2.3 Use of data
- is all data considered

D.2.4 Type of constructs (intermediate level processing) in reasoning process and their complexity
- for support structure

A Taxonomy of Guidelines for Developing Expert System Software in a Military Environment

D.2.5 Amount of support structure for hypothesis
- number of intermediate levels of hypotheses (depth of reasoning chain)
- step size process

D.2.6 Error recovery techniques
- backtracking

D.2.7 Resource allocation
- external (cost model)

D.2.8 Focus of attention (ordering of tasks, agenda, planning)
- for tasks
- for data

D.2.9 Dealing with uncertainty (and propagation of uncertainty)

D.2.10 Stopping criteria

D.3 Knowledge base

D.3.1 Components of knowledge base (form and complexity)
- of inference rules, static knowledge, etc.

E.0 Other Requirements and Considerations for Design and Implementation

E.1 Input/Output
- natural language, graphics, etc.

E.2 Explanation capability
- reasoning process
- history list
- contents of knowledge base
- hypotheses
- data (summary, etc.)

E.3 Mode of operation
- batch vs. interactive
- volunteered information
- user interrupts
- multiple users
- real time

A Taxonomy of Guidelines for Developing Expert System Software in a Military Environment

E.4 Operating environment
- hardware (computers, multiprocessors, etc.)
- operating system
- capability
- size (memory, disk)
- speed

E.5 System requirements and performance
- speed
- size
- memory, disk

E.6 Implementation language

E.7 Representation (data structures used, etc.)

E.8 Methods for validating system performance

F.0 Problems

(Kline and Dolins, 1985 pp 116-119)

APPENDIX G

CAA SKILL ASSESSMENT

G-1. INTRODUCTION. Training for new technology is like shooting at a target with blurred edges. When target behaviors are unclear extensive feedback and tuning must be used to hit the criterion behavior desired from the training (i.e., process control). This appendix reviews the present status of CAA's general skill level, reviews training system design, and draws conclusions about integration of AI technology into the organization.

G-2. PRESENT SKILL LEVELS. CAA has a diverse and highly qualified staff of 222 military and civilian personnel. At the end of Fiscal Year 1985 the approximate technical and educational areas of the CAA staff were as follows:

Table G-1. Technical Speciality and Education of CAA

Technical Area	Military	Civilian
Operations Research	53	57
Military Operations	21	24
Math/Statistics	0	22
Computer Science	2	8
Logistics/Personnel	26	0
Science/Engineering	0	2
Threat	7	0
Total	109	113
Education		
Phd	1%	15%
Masters	22%	37%
Bachelors	22%	37%

The 113 military officers assigned to the CAA technical staff have a 18 to 36 month tours of duty. Their education and training is completed before arrival at CAA. The officers present at CAA at the end of fiscal year 1985 have the following branch specialities.

Table G-2. CAA Officers by Branch

Branch	Number	Group
FA	17	
INF	15	
ADA	12	Combat Arms - 58
AR	12	
AV	2	
ENG	9	
SC	6	
MI	4	Combat Support - 24
CM	3	
MP	2	
AG	9	
TC	6	
OD	4	Combat Service Support - 24
QM	3	
FC	1	
MSC	1	
Total	106	

The present skill level of the CAA staff appears to be varied. The common denominator of the CAA staff is the rapid ability to learn (i.e., high native intelligence) and adapt new knowledge to the study process. For example, the use of microcomputers was integrated into the work program within a few months. The use of Army training system design may decrease the time and effort needed to rapidly adapt AI technology.

G-3. TRAINING SYSTEM DESIGN. Training systems that allow feedback and tuning need to avoid isolation from the work process at CAA. Bryan and Regan (1972) suggested that developing training systems should include:

1. Describing the Job
2. Task Analysis
3. Functional Training Requirements
4. Trade-Off Analysis

This notion of relating training development to the work process was expanded upon by the Army Combat Arms Training Board. The Instructional Systems Development (ISD) model developed by Branson, Rayner, Cox, Furman, King, and Hannom (1975) provided an 18 step sequence for developing interservice training. An agency-wide training effort in AI would appear to profit from the ISD model. The current Army proponent for training development using the ISD model is the Army's Training and Doctrine Command (TRADOC) schools. The use of TRADOC schools in developing a CAA training program for integration of AI technology into the work program would appear to save time, effort, and cost in developing an AI training program. Winston (1985) has suggested that an Army-wide analysis be conducted to assess which schools within the Army currently teach AI.

G-4. CONCLUSION. The staff of CAA appears to be capable of using AI technology in the agencies models, methodologies, and studies. The goal of any training program should be to get as many individuals within the agency started on productive work on the CRT as soon as possible. The training of CAA staff in AI hardware, software, toolkits and methodologies appears to be an evolutionary effort. Training, plus experience working with AI may be necessary to apply the technology to CAA missions. It would appear that an AI training research effort internal to the Army and conducted by either TRADOC schools or an organization like the US Army Research Institute's Training Laboratory would have a great potential pay-off for CAA and the Army. Two factors appear important in developing an AI technology training program at CAA. First, the analysis of the learning task and the definition of the performance objective requires the participation of the management and staff of CAA with the professional training developer. This will allow the organizational goals and objective to be infused into the development process and subsequently into the AI technology training program. Second, an adequate resourcing of the effort is required. AI training is costly and time consuming. It is estimated that a LISP programer using the LISP machine at an expert level takes 24-36 months of training and experience. Haas and Walleck (1986) have suggested that training costs in manfacturing are related to industrial productivity:

> ... the cost of training a skilled Mexican engineer is between $8 and $11 a year. The cost of training a comparably skilled individual in Japan was between $22,000 and $28,000 a year and between $27,000 and $45,000 in the U.S.
>
> (Haas and Walleck, 1986 p 28)

If a similar relationship between training and productivity exists at CAA then it is strongly suggested that CAA budget training costs closer to that of the U.S. and Japan than Mexico for training its professional staff in AI technology.

APPENDIX H

AI DEMO PROJECT: FORCEM COMMAND AND CONTROL

H-1. The screen display and LISP code on the following pages were developed from the Command and Control (C2) logic in the CAA Force Evaluation Model (FORCEM) developed by Metzger (1985). The program was designed by Drs. William S. Faught and Richard B. Modjeski during the Ninth International Joint Conference on Artificial Intelligence, 18-23 August 1985, at the University of California at Los Angeles. Mr. Amos Oshrin, of Intellicorp, coded the project using the Intellicorp Knowledge Engineering Environment (KEE) software. Mr. Ronald Rockwell, Intellicorp, Washington, D.C. was also instrumental in demonstrating the prototype to the staff of CAA and the US Army Artificial Intelligence Center.

H-2. The FORCEM Model is a fully automated theater-level combat simulation model. It is used by CAA as an analytic tool in the study of theater-level issues for the Army. The simulation is time-stepped. Each model cycle represents 12 hours of real time. The model represents the functional areas of communications, command-and-control, division level force attrition, fire support, and combat service support. The basic element in the model is the unit. Units such as Corps or Division Headquarters contain assets that can be destroyed or consumed during the simulation. Units can own subordinate units in a hierarchical structure within the model logic.

H-3. The smallest self-contained unit in the FORCEM Model is the division. The model concepts used to move the division and corps provide a necessary but not sufficient condition for defining command and control logic. Metgzer (1985) summarized the following FORCEM Model concepts:

> A division can move about the battlefield, and engage in combat with enemy divisions. Movement of a division takes account of terrain and intensity of combat, and is controlled through the mechanism of a list of "objective points." At any given time, a division has a list of objective points; it move toward the second point. Movement of a division ceases when the last objective point has been reached....
> The movement of a division subordinate to a corps command and control headquarters is managed through two mechanisms: "corps boundaries" and "corps control phase lines." ... A corps boundary is a continuous piecewise linear curve; two such boundaries constrain the lateral movement of divisions subordinate to the corps.
> Control phase lines are lines orthogonal to, and joining, the corps boundaries. Three such lines exist for the corps at a given time, and contain the most current objective points for subordinate divisions. The first (most forward) phase line contains the most current objective points for those divisions that are

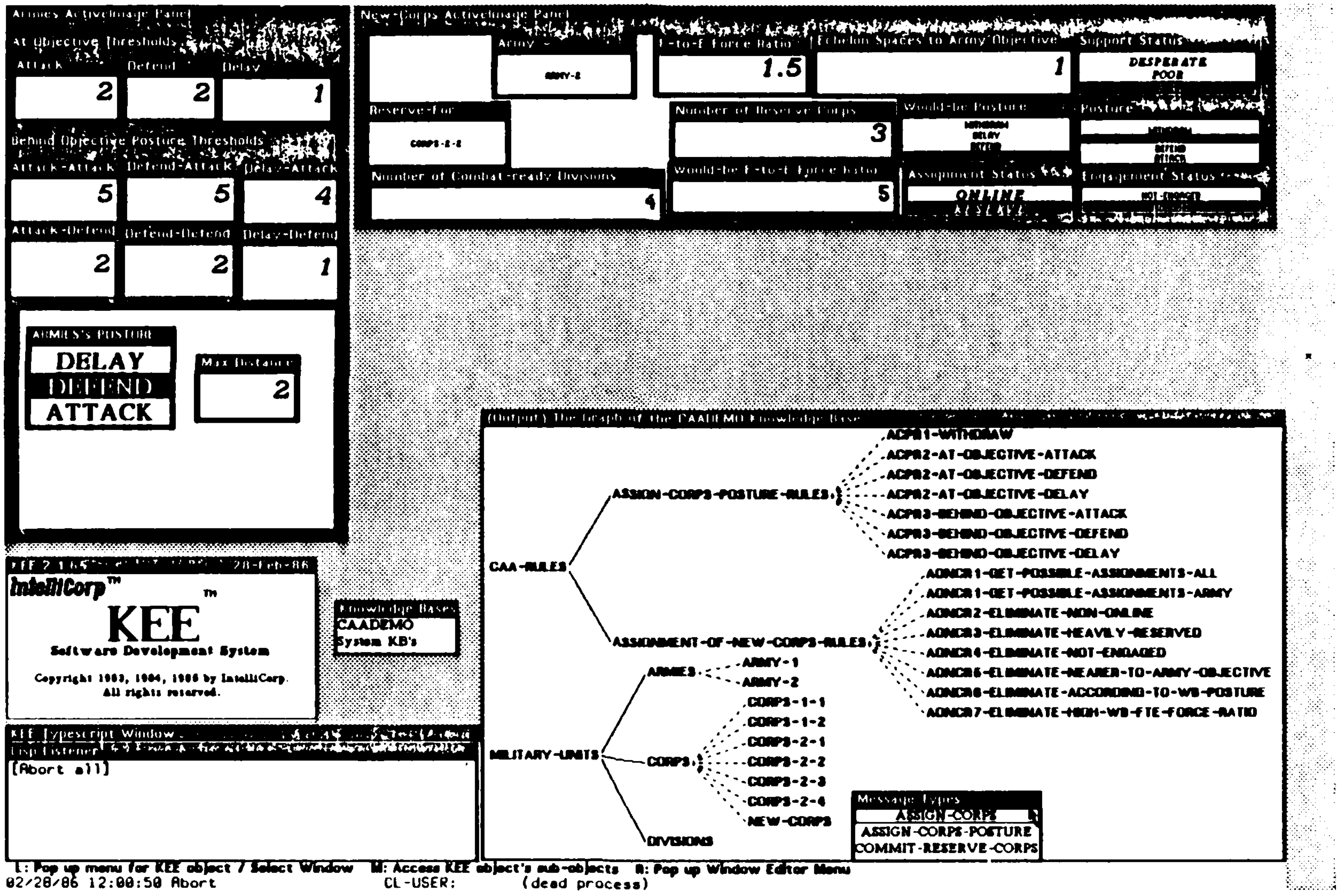
assigned to the first echelon (i.e., committed to combat); the second contains the most current objective points for those divisions that are assigned to the second echelon (i.e., ready for combat but not committed); and the third contains the most current objective points for those divisions; that are assigned to the third echelon (i.e., being refurbished).... The 'organization for combat' of a corps specifies how many subordinate divisions, and which ones, are assigned to each of the three echelons....FORCEM also treats reserve corps'. The movement of the reserve corps is controlled by assigning it the same lateral boundaries as the online corps, and lockstepping the movement (forward or backward) of the control phase lines of the reserve corps to the movement of the control phase lines of the online corps....'Type of operation' is a model parameter applicable to a corps headquarters; it determines conditions that are applied in deciding whether, and in which direction, to move the corps control phase lines....'posture' ...denotes the intensity with which the unit is to pursue its objective. Providing the framework for FORCEM command and control decisionmaking is the concept of an 'operations order' (OPORDER). It contains two kinds of information: directions (e.g., posture) to the given unit from its parent command and control headquarters and priorities for the given unit--as specified by the parent....The model use can influence and direct command and control decisionmaking through two mechanisms: the 'master plan' and the 'knowledge base'....The knowledge base is composed of a set of rules built into the model, and a set of input threshold parameters....Decisions are made in FORCEM by applying the knowledge base and the state of variables in the perceived data base. Because the rules are written into the model the variables in the perceived data base to be accessed for command and control decisions have been preselected.

(Metzger, 1985, pp 2-11)

H-4. The FORCEM command and control demonstration model used the following decision criteria from the CAA FORCEM Model:

a. Assignment of new corps
b. Commitment of reserve corps
c. Corps posture
d. Corps movement

The decision criteria by Metzger (1985) were checked against the program references in the FORCEM Model. The IF-THEN-ELSE statements from the Simscript code were extracted and analyzed against the decision criteria for clarity and completeness. The KEE Rulesystem2 software (version 2.0) was used to specify the FORCEM rule premises in the software and allow centralization of the model reasoning process. In effect this created a KEE knowledge base of FORCEM rules. The object-oriented programing style of using LISP code to call methods through an object by messages was used to tie the FORCEM rules together. The active images function in KEE was used to represent values of the FORCEM command and control knowledge base and to show the relations between objects in the code. For example, the FORCEM Simscript subroutine to assign postures from an Army to a subordinate Corps was taken from subroutine CAC.MISSION.DESIGNATION and reformatted into (ASSIGN-CORPS-POSTURE-RULES) found in section H-3 with monitoring images. An example of the graphic representation and the LISP code for the FORCEM command and control decision logic is given in the following pages. Please note that this rapid prototype was accomplished in 36 hours of coding using the KEE software toolkit on a Symbolics LISP Machine Model 3670.



Armies ActiveImage Panel

At Objective Thresholds

Attack	Defend	Delay
2	2	1

Behind Objective Posture Thresholds

Attack-Attack	Defend-Attack	Delay-Attack
5	5	4

Attack-Defend	Defend-Defend	Delay-Defend
2	2	1

New-Corps ActiveImage Panel

Army	ARMY-2
F-to-E Force Ratio	1.5
Echelon Spaces to Army Objective	1
Support Status	DESPERATE POOR
Reserve-For	CORPS-2-2
Number of Reserve Corps	3
Would-be Posture	WITHDRAW DELAY DEFEND
Posture	WITHDRAW DEFEND ATTACK
Number of Combat-ready Divisions	4
Would-be F-to-E Force Ratio	5
Assignment Status	ONLINE RESERVE
Engagement Status	NOT-ENGAGED

KEE 2.1.65 28-Feb-86

IntelliCorp™

KEE™

Software Development System

Copyright 1983, 1984, 1986 by IntelliCorp. All rights reserved.

(Output) The Graph of the CAADEMO Knowledge Base

```
CAA-RULES
    ASSIGN-CORPS-POSTURE-RULES
        ACPR1-WITHDRAW
        ACPR2-AT-OBJECTIVE-ATTACK
        ACPR2-AT-OBJECTIVE-DEFEND
        ACPR2-AT-OBJECTIVE-DELAY
        ACPR3-BEHIND-OBJECTIVE-ATTACK
        ACPR3-BEHIND-OBJECTIVE-DEFEND
        ACPR3-BEHIND-OBJECTIVE-DELAY
    ASSIGNMENT-OF-NEW-CORPS-RULES
        AONCR1-GET-POSSIBLE-ASSIGNMENTS-ALL
        AONCR1-GET-POSSIBLE-ASSIGNMENTS-ARMY
        AONCR2-ELIMINATE-NON-ONLINE
        AONCR3-ELIMINATE-HEAVILY-RESERVED
        AONCR4-ELIMINATE-NOT-ENGAGED
        AONCR5-ELIMINATE-NEARER-TO-ARMY-OBJECTIVE
        AONCR6-ELIMINATE-ACCORDING-TO-WB-POSTURE
        AONCR7-ELIMINATE-HIGH-WB-FTE-FORCE-RATIO
MILITARY-UNITS
    ARMIES
        ARMY-1
        ARMY-2
    CORPS
        CORPS-1-1
        CORPS-1-2
        CORPS-2-1
        CORPS-2-2
        CORPS-2-3
        CORPS-2-4
        NEW-CORPS
    DIVISIONS
```

KEE Typescript Window

Lisp Listener

```
[Abort all]
```

```
L: Pop up menu for KEE object / Select Window   M: Access KEE object's sub-objects   R: Pop up Window Editor Menu
02/28/86 12:00:50 Abort                CL-USER:          (dead process)
```

```
;;; -*- Mode:LISP; Package:KEE; Base:10. -*-

(CAADEMO
  ("" "23-Nov-85 17:57:49" "AUGUSTE" "23-Nov-85 19:48:06")
  NIL
  (KNOWLEDGEBASES)
  NIL
  ()
  ((KBCOPY ((BEFORE (BEFOREKBCOPYIMAGE SELF KB NEWKB)) (AFTER (AFTERKBCOPYIMAGE SELF
                                                                                KB
                                                                                NEWKB)))
           METHOD
           METHOD
           NIL
           NIL)
   (KBDELETE ((BEFORE (BEFOREKBDELETEIMAGE SELF KB.OR.REF NOASKP NOERRORP)))
             METHOD
             METHOD
             NIL
             NIL)
   (KBLOAD ((AFTER (AFTERKBLOADIMAGE SELF KBSTREAM QUIETP NORELINKP KB)))
           METHOD
           METHOD
           NIL
           NIL)
   (KBMETHODFILE (CAADEMO))
   (KBSAVE ((BEFORE (BEFOREKBSAVEIMAGE SELF KB QUIETP MAKEMODE NEWDIRECTORY)))
           METHOD
           METHOD
           NIL
           NIL)
   (KBSIZE 59)
   (KEE.DEVELOPMENT.VERSION.NUMBER 0)
   (KEE.MAJOR.VERSION.NUMBER 2)
   (KEE.MINOR.VERSION.NUMBER 1)
   (KEE.PATCH.VERSION.NUMBER 3)
   (KEEVERSION KEE2.1)))


(ACPR1-WITHDRAW
  ("AUGUSTE" "24-Nov-85 1:10:16")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#Wff (A POSTURE OF ?CORPS IS WITHDRAW)))
   (EXTERNAL.FORM (IF (< (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                      THEN
                      (THE POSTURE OF ?CORPS IS WITHDRAW)))
   (PREMISE (#Wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS IS ?VAR35) #Wff (< ?VAR35 0)))))


(ACPR2-AT-OBJECTIVE-ATTACK
  ("AUGUSTE" "24-Nov-85 1:34:10")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#Wff (A POSTURE OF ?CORPS IS ?VAR44)))
   (EXTERNAL.FORM (IF (AND (= (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= ATTACK (THE POSTURE OF ?ARMY))
                           (= ?THRESHOLD (THE AT-OBJECTIVE-ATTACK-POSTURE-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE OF ?CORPS IS (IF (> ?RATIO ?THRESHOLD) 'DEFEND 'DELAY))))
   (PREMISE
    (#Wff ((AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) = 0)
     #Wff (AN ARMY OF ?CORPS IS ?ARMY)
     #Wff (ATTACK = (A POSTURE OF ?ARMY))
     #Wff (?THRESHOLD = (AN AT-OBJECTIVE-ATTACK-POSTURE-THRESHOLD OF ?ARMY))
     #Wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
     #Wff (?VAR45 = (> ?RATIO ?THRESHOLD))
     #Wff (?VAR44 = (IF ?VAR45 DEFEND DELAY))))))
```

```
(ACPR2-AT-OBJECTIVE-DEFEND
  ("AUGUSTE" "24-Nov-85 2:29:18")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#wff (A POSTURE OF ?CORPS IS ?VAR48)))
   (EXTERNAL.FORM (IF (AND (= (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= DEFEND (THE POSTURE OF ?ARMY))
                           (= ?THRESHOLD (THE AT-OBJECTIVE-DEFEND-POSTURE-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE OF ?CORPS IS (IF (> ?RATIO ?THRESHOLD) 'DEFEND 'DELAY))))
   (PREMISE
    (#wff ((AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) = 0)
     #wff (AN ARMY OF ?CORPS IS ?ARMY)
     #wff (DEFEND = (A POSTURE OF ?ARMY))
     #wff (?THRESHOLD = (AN AT-OBJECTIVE-DEFEND-POSTURE-THRESHOLD OF ?ARMY))
     #wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
     #wff (?VAR49 = (> ?RATIO ?THRESHOLD))
     #wff (?VAR48 = (IF ?VAR49 DEFEND DELAY))))))


(ACPR2-AT-OBJECTIVE-DELAY
  ("AUGUSTE" "24-Nov-85 2:29:25")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#wff (A POSTURE OF ?CORPS IS ?VAR52)))
   (EXTERNAL.FORM (IF (AND (= (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= DELAY (THE POSTURE OF ?ARMY))
                           (= ?THRESHOLD (THE AT-OBJECTIVE-DELAY-POSTURE-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE OF ?CORPS IS (IF (> ?RATIO ?THRESHOLD) 'DEFEND 'DELAY))))
   (PREMISE
    (#wff ((AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) = 0)
     #wff (AN ARMY OF ?CORPS IS ?ARMY)
     #wff (DELAY = (A POSTURE OF ?ARMY))
     #wff (?THRESHOLD = (AN AT-OBJECTIVE-DELAY-POSTURE-THRESHOLD OF ?ARMY))
     #wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
     #wff (?VAR53 = (> ?RATIO ?THRESHOLD))
     #wff (?VAR52 = (IF ?VAR53 DEFEND DELAY))))))


(ACPR3-BEHIND-OBJECTIVE-ATTACK
  ("AUGUSTE" "24-Nov-85 2:46:53")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#wff (A POSTURE OF ?CORPS IS ?VAR99)))
   (EXTERNAL.FORM (IF (AND (> (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= ATTACK (THE POSTURE OF ?ARMY))
                           (= ?A-THRESHOLD
                              (THE BEHIND-OBJECTIVE-ATTACK-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
                           (= ?D-THRESHOLD
                              (THE BEHIND-OBJECTIVE-ATTACK-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE
                           OF
                           ?CORPS
                           IS
                           (IF (> ?RATIO ?A-THRESHOLD)
                               'ATTACK
                               (IF (> ?RATIO ?D-THRESHOLD) 'DEFEND 'DELAY)))))
   (PREMISE
    (#wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS IS ?VAR98)
     #wff (> ?VAR98 0)
     #wff (AN ARMY OF ?CORPS IS ?ARMY)
     #wff (ATTACK = (A POSTURE OF ?ARMY))
```

```
      #Wff (?A-THRESHOLD = (A BEHIND-OBJECTIVE-ATTACK-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
      #Wff (?D-THRESHOLD = (A BEHIND-OBJECTIVE-ATTACK-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
      #Wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
      #Wff (?VAR102 = (> ?RATIO ?D-THRESHOLD))
      #Wff (?VAR101 = (IF ?VAR102 DEFEND DELAY))
      #Wff (?VAR100 = (> ?RATIO ?A-THRESHOLD))
      #Wff (?VAR99 = (IF ?VAR100 ATTACK ?VAR101))))))


(ACPR3-BEHIND-OBJECTIVE-DEFEND
  ("AUGUSTE" "24-Nov-85 2:47:00")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#Wff (A POSTURE OF ?CORPS IS ?VAR109)))
   (EXTERNAL.FORM (IF (AND (> (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= DEFEND (THE POSTURE OF ?ARMY))
                           (= ?A-THRESHOLD
                              (THE BEHIND-OBJECTIVE-DEFEND-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
                           (= ?D-THRESHOLD
                              (THE BEHIND-OBJECTIVE-DEFEND-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE
                           OF
                           ?CORPS
                           IS
                           (IF (> ?RATIO ?A-THRESHOLD)
                               'ATTACK
                               (IF (> ?RATIO ?D-THRESHOLD)' 'DEFEND 'DELAY)))))
   (PREMISE
    (#Wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS IS ?VAR108)
     #Wff (> ?VAR108 0)
     #Wff (AN ARMY OF ?CORPS IS ?ARMY)
     #Wff (DEFEND = (A POSTURE OF ?ARMY))
     #Wff (?A-THRESHOLD = (A BEHIND-OBJECTIVE-DEFEND-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
     #Wff (?D-THRESHOLD = (A BEHIND-OBJECTIVE-DEFEND-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
     #Wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
     #Wff (?VAR112 = (> ?RATIO ?D-THRESHOLD))
     #Wff (?VAR111 = (IF ?VAR112 DEFEND DELAY))
     #Wff (?VAR110 = (> ?RATIO ?A-THRESHOLD))
     #Wff (?VAR109 = (IF ?VAR110 ATTACK ?VAR111))))))


(ACPR3-BEHIND-OBJECTIVE-DELAY
  ("AUGUSTE" "24-Nov-85 2:47:03")
  NIL
  (ASSIGN-CORPS-POSTURE-RULES)
  NIL
  ()
  ((ASSERTION (#Wff (A POSTURE OF ?CORPS IS ?VAR119)))
   (EXTERNAL.FORM (IF (AND (> (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS) 0)
                           (THE ARMY OF ?CORPS IS ?ARMY)
                           (= DELAY (THE POSTURE OF ?ARMY))
                           (= ?A-THRESHOLD
                              (THE BEHIND-OBJECTIVE-DELAY-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
                           (= ?D-THRESHOLD
                              (THE BEHIND-OBJECTIVE-DELAY-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
                           (= ?RATIO (THE F-TO-E-FORCE-RATIO OF ?CORPS)))
                      THEN
                      (THE POSTURE
                           OF
                           ?CORPS
                           IS
                           (IF (> ?RATIO ?A-THRESHOLD)
                               'ATTACK
                               (IF (> ?RATIO ?D-THRESHOLD) 'DEFEND 'DELAY)))))
   (PREMISE
    (#Wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS IS ?VAR118)
     #Wff (> ?VAR118 0)
     #Wff (AN ARMY OF ?CORPS IS ?ARMY)
     #Wff (DELAY = (A POSTURE OF ?ARMY))
     #Wff (?A-THRESHOLD = (A BEHIND-OBJECTIVE-DELAY-POSTURE-ATTACK-THRESHOLD OF ?ARMY))
     #Wff (?D-THRESHOLD = (A BEHIND-OBJECTIVE-DELAY-POSTURE-DEFEND-THRESHOLD OF ?ARMY))
```

```
        #Wff (?RATIO = (A F-TO-E-FORCE-RATIO OF ?CORPS))
        #Wff (?VAR122 = (> ?RATIO ?D-THRESHOLD))
        #Wff (?VAR121 = (IF ?VAR122 DEFEND DELAY))
        #Wff (?VAR120 = (> ?RATIO ?A-THRESHOLD))
        #Wff (?VAR119 = (IF ?VAR120 ATTACK ?VAR121))))))


(AONCR1-GET-POSSIBLE-ASSIGNMENTS-ALL
  ("AUGUSTE" "23-Nov-85 20:26:58")
  NIL
  (ASSIGNMENT-OF-NEW-CORPS-RULES)
  NIL
  ()
  ((ACTION
    ((PUT.VALUES ?CORPS 'POSSIBLE-RESERVE-ASSIGNMENT (UNIT.ALLCHILDREN 'CORPS 'MEMBER))))
   (ASSERTION NIL)
   (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                           (NOT (QUERY '(THE ARMY OF ,?CORPS IS ?ARMY))))
                      DO
                      (PUT.VALUES ?CORPS
                                  'POSSIBLE-RESERVE-ASSIGNMENT
                                  (UNIT.ALLCHILDREN 'CORPS 'MEMBER))))
   (PREMISE
    (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
     #Wff (?VAR13 = (SI:XR-BQ-LIST* THE ARMY OF ?CORPS (QUOTE (IS ?ARMY))))
     #Wff (NOT (QUERY ?VAR13))))
   (WEIGHT (1))))


(AONCR1-GET-POSSIBLE-ASSIGNMENTS-ARMY
  ("AUGUSTE" "23-Nov-85 20:26:59")
  NIL
  (ASSIGNMENT-OF-NEW-CORPS-RULES)
  NIL
  ()
  ((ACTION ((PUT.VALUES ?CORPS 'POSSIBLE-RESERVE-ASSIGNMENT (GET.VALUES ?ARMY 'OWNS))))
   (ASSERTION NIL)
   (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                           (THE ARMY OF ?CORPS IS ?ARMY))
                      DO
                      (PUT.VALUES ?CORPS
                                  'POSSIBLE-RESERVE-ASSIGNMENT
                                  (GET.VALUES ?ARMY 'OWNS))))
   (PREMISE
    (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS) #Wff (AN ARMY OF ?CORPS IS ?ARMY)))
   (WEIGHT (1))))


(AONCR2-ELIMINATE-NON-ONLINE
  ("" "23-Nov-85 19:23:26")
  NIL
  (ASSIGNMENT-OF-NEW-CORPS-RULES)
  NIL
  ()
  ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2)) ))
   (ASSERTION NIL)
   (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                           (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                           (NOT (= ONLINE (THE ASSIGNMENT-STATUS OF ?CORPS2))))
                      DO
                      (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2))))
   (PREMISE
    (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
     #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
     #Wff (NOT (ONLINE = (AN ASSIGNMENT-STATUS OF ?CORPS2)))))
   (WEIGHT (2))))


(AONCR3-ELIMINATE-HEAVILY-RESERVED
  ("" "23-Nov-85 19:25:22")
  NIL
  (ASSIGNMENT-OF-NEW-CORPS-RULES)
  NIL
  ()
  ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2)) ))
   (ASSERTION NIL)
  (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
                          (NOT (= ?CORPS2 ?CORPS3))
                          (> (THE NUMBER-OF-RESERVE-CORPS OF ?CORPS2)
                             (THE NUMBER-OF-RESERVE-CORPS OF ?CORPS3)))
                     DO
                     (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2))))
  (PREMISE
   (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
    #Wff (NOT (?CORPS2 = ?CORPS3))
    #Wff (A NUMBER-OF-RESERVE-CORPS OF ?CORPS3 IS ?VAR11)
    #Wff (A NUMBER-OF-RESERVE-CORPS OF ?CORPS2 IS ?VAR10)
    #Wff (> ?VAR10 ?VAR11)))
  (WEIGHT (3))))


(AONCR4-ELIMINATE-NOT-ENGAGED
 ("" "23-Nov-85 19:27:11")
 NIL
 (ASSIGNMENT-OF-NEW-CORPS-RULES)
 NIL
 ()
 ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS3)) ))
  (ASSERTION NIL)
  (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
                          (NOT (= ?CORPS2 ?CORPS3))
                          (= ENGAGED (THE ENGAGEMENT-STATUS OF ?CORPS2))
                          (= NOT-ENGAGED (THE ENGAGEMENT-STATUS OF ?CORPS3)))
                     DO
                     (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS3))))
  (PREMISE
   (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
    #Wff (NOT (?CORPS2 = ?CORPS3))
    #Wff (ENGAGED = (AN ENGAGEMENT-STATUS OF ?CORPS2))
    #Wff (NOT-ENGAGED = (AN ENGAGEMENT-STATUS OF ?CORPS3))))
  (WEIGHT (4))))


(AONCR5-ELIMINATE-NEARER-TO-ARMY-OBJECTIVE
 ("" "23-Nov-85 19:31:39")
 NIL
 (ASSIGNMENT-OF-NEW-CORPS-RULES)
 NIL
 ()
 ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2)) ))
  (ASSERTION NIL)
  (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                          (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
                          (NOT (= ?CORPS2 ?CORPS3))
                          (< (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS2)
                             (THE ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS3)))
                     DO
                     (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2))))
  (PREMISE
   (#Wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
    #Wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
    #Wff (NOT (?CORPS2 = ?CORPS3))
    #Wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS3 IS ?VAR21)
    #Wff (AN ECHELON-SPACES-TO-ARMY-OBJECTIVE OF ?CORPS2 IS ?VAR20)
    #Wff (< ?VAR20 ?VAR21)))
  (WEIGHT (5))))


(AONCR6-ELIMINATE-ACCORDING-TO-WB-POSTURE
 ("AUGUSTE" "23-Nov-85 21:21:00")
 NIL
 (ASSIGNMENT-OF-NEW-CORPS-RULES)
```

```
  NIL
  ()
  ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS3)) ))
   (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                           (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                           (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
                           (NOT (= ?CORPS2 ?CORPS3))
                           (ALPHALESSP (THE WOULD-BE-POSTURE OF ?CORPS2)
                                       (THE WOULD-BE-POSTURE OF ?CORPS3)))
                      DO
                      (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS3))))
   (PREMISE
    (#wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
     #wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
     #wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
     #wff (NOT (?CORPS2 = ?CORPS3))
     #wff (A WOULD-BE-POSTURE OF ?CORPS3 IS ?VAR33)
     #wff (A WOULD-BE-POSTURE OF ?CORPS2 IS ?VAR32)
     #wff (ALPHALESSP ?VAR32 ?VAR33)))
   (WEIGHT (6))))


(AONCR7-ELIMINATE-HIGH-WB-FTE-FORCE-RATIO
  ("AUGUSTE" "23-Nov-85 21:21:03")
  NIL
  (ASSIGNMENT-OF-NEW-CORPS-RULES)
  NIL
  ()
  ((ACTION ((RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2)) ))
   (EXTERNAL.FORM (IF (AND (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
                           (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
                           (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
                           (NOT (= ?CORPS2 ?CORPS3))
                           (> (THE WOULD-BE-F-TO-E-FORCE-RATIO OF ?CORPS2)
                              (THE WOULD-BE-F-TO-E-FORCE-RATIO OF ?CORPS3)))
                      DO
                      (RETRACT '(A POSSIBLE-RESERVE-ASSIGNMENT OF ,?CORPS IS ,?CORPS2))))
   (PREMISE
    (#wff (THE CORPS-TO-BE-ASSIGNED OF CORPS IS ?CORPS)
     #wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS2)
     #wff (A POSSIBLE-RESERVE-ASSIGNMENT OF ?CORPS IS ?CORPS3)
     #wff (NOT (?CORPS2 = ?CORPS3))
     #wff (A WOULD-BE-F-TO-E-FORCE-RATIO OF ?CORPS3 IS ?VAR29)
     #wff (A WOULD-BE-F-TO-E-FORCE-RATIO OF ?CORPS2 IS ?VAR28)
     #wff (> ?VAR28 ?VAR29)))
   (WEIGHT (7))))


(ARMIES
  ("" "23-Nov-85 18:02:09")
  (MILITARY-UNITS)
  ((CLASSES GENERICUNITS))
  NIL
  ((AT-OBJECTIVE-ATTACK-POSTURE-THRESHOLD (2) NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
   (AT-OBJECTIVE-DEFEND-POSTURE-THRESHOLD (2) NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
   (AT-OBJECTIVE-DELAY-POSTURE-THRESHOLD (1) NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-ATTACK-POSTURE-ATTACK-THRESHOLD (5)
                                                     NIL
                                                     (NUMBER)
                                                     NIL
                                                     ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-ATTACK-POSTURE-DEFEND-THRESHOLD (2)
                                                     NIL
                                                     (NUMBER)
                                                     NIL
                                                     ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-DEFEND-POSTURE-ATTACK-THRESHOLD (5)
                                                     NIL
                                                     (NUMBER)
                                                     NIL
                                                     ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-DEFEND-POSTURE-DEFEND-THRESHOLD (2)
                                                     NIL
                                                     (NUMBER)
                                                     NIL
                                                     ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-DELAY-POSTURE-ATTACK-THRESHOLD (4)
                                                    NIL
                                                    (NUMBER)
                                                    NIL
                                                    ((CARDINALITY.MAX (1))))
   (BEHIND-OBJECTIVE-DELAY-POSTURE-DEFEND-THRESHOLD (1)
                                                    NIL
                                                    (NUMBER)
                                                    NIL
                                                    ((CARDINALITY.MAX (1))))
   (MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS (2) NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
   (OBJECTIVE-PHASE-LINE NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
   (OWNS NIL NIL (CORPS))
   (POSTURE (DEFEND) NIL ((ONE.OF ATTACK DEFEND DELAY))))
  ((\IMAGE.PANEL (#Unit (IMAGE.PANEL01709 CAADEMO) #Unit (IMAGE.PANEL01708 CAADEMO)))))


(ARMY-1
  ("AUGUSTE" "23-Nov-85 20:33:26")
  NIL
  (ARMIES)
  NIL
  ()
  ((OWNS (CORPS-1-1 CORPS-1-2))))


(ARMY-2
  ("AUGUSTE" "23-Nov-85 20:33:28")
  NIL
  (ARMIES)
  NIL
  ()
  ((OWNS (CORPS-2-1 CORPS-2-2 CORPS-2-4 CORPS-2-3))))


(ASSIGN-CORPS-POSTURE-RULES
  ("AUGUSTE" "24-Nov-85 0:55:21")
  (CAA-RULES (RULES RULESYSTEM2))
  ((RULE.CLASSES RULESYSTEM2))
  NIL
  ((ASSERTION)
   (EXTERNAL.FORM)
   (PREMISE))
  ((AND.TRACE)
   (FC.TRACE)
   (IMMEDIATE.RULE.APPLICATION.MODE)
   (OR.TRACE)
   (PANEL.TRACE)
   (STEPPER.MODE)
   (TEXT.TRACE)))


(ASSIGNMENT-OF-NEW-CORPS-RULES
  ("" "23-Nov-85 19:19:19")
  (CAA-RULES (RULES RULESYSTEM2))
  ((RULE.CLASSES RULESYSTEM2))
  NIL
  ((ASSERTION)
   (EXTERNAL.FORM)
   (PREMISE)
   (WEIGHT NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1)))))
  ((AND.TRACE (OFF))
   (FC.TRACE (OFF))
   (IMMEDIATE.RULE.APPLICATION.MODE)
   (OR.TRACE (OFF))
   (PANEL.TRACE)
   (RESOLVE.CONFLICT LEAST.WEIGHT)
   (STEPPER.MODE (OFF))
   (TEXT.TRACE (OFF))))


(CAA-RULES
  ("" "23-Nov-85 19:21:24")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  NIL
```

```
    ()
    ())


  (CORPS
    ("" "23-Nov-85 18:02:09")
    (MILITARY-UNITS)
    ((CLASSES GENERICUNITS))
    NIL
    ((ARMY NIL NIL (ARMIES) NIL ((CARDINALITY.MAX (1))))
     (ASSIGN-CORPS (LAMBDA (SELF &AUX RESERVE-FOR)
                     (PUT.VALUE 'CORPS 'CORPS-TO-BE-ASSIGNED SELF)
                     (FORWARD.CHAIN 'ASSIGNMENT-OF-NEW-CORPS-RULES)
                     (PUT.VALUE SELF 'ASSIGNMENT-STATUS 'RESERVE)
                     (SETQ RESERVE-FOR (GET.VALUE SELF 'POSSIBLE-RESERVE-ASSIGNMENT))
                     (PUT.VALUE SELF 'RESERVE-FOR RESERVE-FOR)
                     (PUT.VALUE SELF 'ARMY (GET.VALUE RESERVE-FOR 'ARMY)))
                   METHOD
                   (METHOD))
     (ASSIGN-CORPS-POSTURE (LAMBDA (SELF)
                             (REMOVE.ALL.LOCAL.VALUES SELF 'POSTURE)
                             (QUERY '(THE POSTURE OF ,SELF IS ?POSTURE)
                                    1
                                    'ASSIGN-CORPS-POSTURE-RULES))
                           METHOD
                           (METHOD))
     (ASSIGNMENT-STATUS NIL NIL ((ONE.OF RESERVE ONLINE)) NIL ((CARDINALITY.MAX (1))))
     (COMMIT-RESERVE-CORPS
      (LAMBDA
       (SELF)
       (PROG
        ((ASSIGNMENT-STATUS (GET.VALUE SELF 'ASSIGNMENT-STATUS))
         (SUPPORT-STATUS (GET.VALUE SELF 'SUPPORT-STATUS))
         (NUMBER-OF-COMBAT-READY-DIVISIONS (GET.VALUE SELF 'NUMBER-OF-COMBAT-READY-DIVISIONS))
         (MIN-NUMBER-OF-COMBAT-READY-DIVISIONS
          (GET.VALUE SELF 'MIN-NUMBER-OF-COMBAT-READY-DIVISIONS))
         (ARMY (GET.VALUE SELF 'ARMY))
         MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS
         ONLINE-CORPS
         OC-SPACES-TO-ARMY-OBJECTIVE
         ONLINE-CORPS-POSTURE
         NEIGHBORS
         COMMITTED-TO)
        (REMOVE.ALL.LOCAL.VALUES SELF 'COMMITTED-TO)
        (COND ((NOT (EQUAL ASSIGNMENT-STATUS 'RESERVE))
               (PRINTOUT T T "This is not a reserve unit !!")
               (RETURN 'FAIL-1))
              ((NOT (EQUAL SUPPORT-STATUS 'OKAY))
               (PRINTOUT T T "The support status of this unit is not 'okay !!")
               (RETURN 'FAIL-2))
              ((< NUMBER-OF-COMBAT-READY-DIVISIONS MIN-NUMBER-OF-COMBAT-READY-DIVISIONS)
               (PRINTOUT T T "There are not enough combat ready divisions in this corps !!")
               (RETURN 'FAIL-3)))
        (SETQ
         MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS
         (GET.VALUE ARMY 'MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS))
        (SETQ ONLINE-CORPS (GET.VALUE SELF 'RESERVE-FOR))
        (SETQ OC-SPACES-TO-ARMY-OBJECTIVE (GET.VALUE ONLINE-CORPS
                                                     'ECHELON-SPACES-TO-ARMY-OBJECTIVE))
        (SETQ ONLINE-CORPS-POSTURE (GET.VALUE ONLINE-CORPS 'POSTURE))
        (SETQ NEIGHBORS (GET.VALUES ONLINE-CORPS 'NEIGHBOR))
        (COND ((AND (> OC-SPACES-TO-ARMY-OBJECTIVE 0)
                    (NOT (EQUAL 'ATTACK ONLINE-CORPS-POSTURE))
                    (LOOP FOR
                          NEIGHBOR
                          IN
                          NEIGHBORS
                          WHEN
                          (> (- OC-SPACES-TO-ARMY-OBJECTIVE
                                (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE))
                             MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS)
                          COLLECT
                          NEIGHBOR))
               (PUT.VALUE SELF 'COMMITTED-TO ONLINE-CORPS)
               (PRINTOUT T T "Committed to " (UNIT.NAME ONLINE-CORPS))
               (RETURN ONLINE-CORPS)))
        (COND
         ((SETQ
           COMMITTED-TO
           (CAR (LOOP FOR
                      NEIGHBOR
                      IN
                      NEIGHBORS
                      WHEN
                      (AND (> (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE) 0)
                           (NOT (EQUAL 'ATTACK (GET.VALUE NEIGHBOR 'POSTURE)))
                           (> (- (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE)
                                 OC-SPACES-TO-ARMY-OBJECTIVE)
                              MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS))
                      COLLECT
                      NEIGHBOR)))
          (PUT.VALUE SELF 'COMMITTED-TO COMMITTED-TO)
          (PRINTOUT T T "Committed to " (UNIT.NAME COMMITTED-TO))
          (RETURN COMMITTED-TO)))
        (PRINTOUT T T "Guess nobody needs my help !!")
        (RETURN 'FAIL-5)))
      METHOD
      (METHOD)
      NIL
      NIL)
     (ECHELON-SPACES-TO-ARMY-OBJECTIVE NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (ENGAGEMENT-STATUS NIL NIL ((ONE.OF ENGAGED NOT-ENGAGED)) NIL ((CARDINALITY.MAX (1))))
     (F-TO-E-FORCE-RATIO NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (MIN-NUMBER-OF-COMBAT-READY-DIVISIONS (3) NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (NEIGHBOR NIL NIL (CORPS) NIL ((CARDINALITY.MAX (2))))
     (NUMBER-OF-COMBAT-READY-DIVISIONS NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (NUMBER-OF-RESERVE-CORPS NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (OWNS NIL NIL (DIVISIONS))
     (POSSIBLE-RESERVE-ASSIGNMENT NIL NIL (CORPS))
     (RESERVE-FOR NIL NIL (CORPS) NIL ((CARDINALITY.MAX (1))))
     (SUPPORT-STATUS NIL NIL ((ONE.OF OKAY POOR DESPERATE)) NIL ((CARDINALITY.MAX (1))))
     (WOULD-BE-F-TO-E-FORCE-RATIO NIL NIL (NUMBER) NIL ((CARDINALITY.MAX (1))))
     (WOULD-BE-POSTURE NIL
                       NIL
                       ((ONE.OF ATTACK DEFEND DELAY WITHDRAW))
                       NIL
                       ((CARDINALITY.MAX (1)))))
    ((ASSIGN-CORPS-POSTURES (LAMBDA (SELF &AUX (CORPS (UNIT.ALLCHILDREN SELF 'MEMBER)))
                              (LOOP FOR CORP IN CORPS DO (UNITMSG CORP 'ASSIGN-CORPS-POSTURE)))
                            METHOD
                            (METHOD))
     (CORPS-TO-BE-ASSIGNED (#Unit (NEW-CORPS CAADEMO)) NIL (CORPS) NIL ((CARDINALITY.MAX (1))))))


(CORPS-1-1
  ("AUGUSTE" "23-Nov-85 20:34:29")
  NIL
  (CORPS)
  NIL
  ()
  ((ARMY (ARMY-1))
   (ASSIGNMENT-STATUS (ONLINE))
   (ECHELON-SPACES-TO-ARMY-OBJECTIVE (3))
   (ENGAGEMENT-STATUS (ENGAGED))
   (NEIGHBOR (CORPS-1-2))
   (NUMBER-OF-RESERVE-CORPS (3))
   (WOULD-BE-F-TO-E-FORCE-RATIO (5))
   (WOULD-BE-POSTURE (ATTACK))))


(CORPS-1-2
  ("AUGUSTE" "23-Nov-85 20:34:29")
  NIL
  (CORPS)
  NIL
  ()
  ((ARMY (ARMY-1))
   (ASSIGNMENT-STATUS (ONLINE))
   (ECHELON-SPACES-TO-ARMY-OBJECTIVE (4))
   (ENGAGEMENT-STATUS (ENGAGED))
   (NEIGHBOR (CORPS-1-1))
   (NUMBER-OF-RESERVE-CORPS (3))
  (WOULD-BE-F-TO-E-FORCE-RATIO (5))
  (WOULD-BE-POSTURE (WITHDRAW))))


(CORPS-2-1
 ("AUGUSTE" "23-Nov-85 20:34:29")
 NIL
 (CORPS)
 NIL
 ()
 ((ARMY (ARMY-2))
  (ASSIGNMENT-STATUS (ONLINE))
  (ECHELON-SPACES-TO-ARMY-OBJECTIVE (4))
  (ENGAGEMENT-STATUS (ENGAGED))
  (NUMBER-OF-RESERVE-CORPS (3))
  (WOULD-BE-F-TO-E-FORCE-RATIO (10))
  (WOULD-BE-POSTURE (ATTACK))))


(CORPS-2-2
 ("AUGUSTE" "23-Nov-85 20:34:30")
 NIL
 (CORPS)
 NIL
 ()
 ((ARMY (ARMY-2))
  (ASSIGNMENT-STATUS (ONLINE))
  (ECHELON-SPACES-TO-ARMY-OBJECTIVE (4))
  (ENGAGEMENT-STATUS (ENGAGED))
  (NEIGHBOR (CORPS-2-1 CORPS-2-3))
  (NUMBER-OF-RESERVE-CORPS (3))
  (WOULD-BE-F-TO-E-FORCE-RATIO (5))
  (WOULD-BE-POSTURE (ATTACK))))


(CORPS-2-3
 ("AUGUSTE" "23-Nov-85 22:45:00")
 NIL
 (CORPS)
 NIL
 ()
 ((ARMY (ARMY-2))
  (ASSIGNMENT-STATUS (ONLINE))
  (ECHELON-SPACES-TO-ARMY-OBJECTIVE (1))
  (ENGAGEMENT-STATUS (ENGAGED))
  (NUMBER-OF-RESERVE-CORPS (5))
  (WOULD-BE-F-TO-E-FORCE-RATIO (5))
  (WOULD-BE-POSTURE (ATTACK))))


(CORPS-2-4
 ("AUGUSTE" "23-Nov-85 22:48:32")
 NIL
 (CORPS)
 NIL
 ()
 ((ARMY (ARMY-2))
  (ASSIGNMENT-STATUS (ONLINE))
  (ECHELON-SPACES-TO-ARMY-OBJECTIVE (4))
  (ENGAGEMENT-STATUS (ENGAGED))
  (NUMBER-OF-RESERVE-CORPS (3))
  (WOULD-BE-F-TO-E-FORCE-RATIO (5))
  (WOULD-BE-POSTURE (ATTACK))))


(DIGIACTUATOR01651
 ("AUGUSTE" "24-Nov-85 0:03:05")
 NIL
 ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:METSI)
  (HEIGHT 57)
  (IMAGE.WAS.PAINTED NIL)
  (OBJECT.DISPLAYED #Slot (AT-OBJECTIVE-ATTACK-POSTURE-THRESHOLD ARMIES CAADEMO MEMBER))
```

```
   (REGION (785 198 97 57)
           NIL
           NIL
           NIL
           ((ICON.REGION (734 270 357 48)) (EXPAND.REGION (785 198 97 57))))
   (SAVED.VALUES 2)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01709 CAADEMO))
   (TITLE "Attack")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 97)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR01678
  ("AUGUSTE" "24-Nov-85 0:04:30")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 56)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (AT-OBJECTIVE-DEFEND-POSTURE-THRESHOLD ARMIES CAADEMO MEMBER))
   (REGION (881 199 84 56)
           NIL
           NIL
           NIL
           ((ICON.REGION (734 246 357 48)) (EXPAND.REGION (881 199 84 56))))
   (SAVED.VALUES 2)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01709 CAADEMO))
   (TITLE "Defend")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 84)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR01679
  ("AUGUSTE" "24-Nov-85 0:04:48")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 56)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (AT-OBJECTIVE-DELAY-POSTURE-THRESHOLD ARMIES CAADEMO MEMBER))
   (REGION (966 199 84 56)
           NIL
           NIL
           NIL
           ((ICON.REGION (741 246 350 48)) (EXPAND.REGION (966 199 84 56))))
   (SAVED.VALUES 1)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01709 CAADEMO))
   (TITLE "Delay")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 84)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
```

```
                ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))

(DIGIACTUATOR01741
  ("AUGUSTE" "24-Nov-85 0:24:27")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 59)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED
    #Slot (BEHIND-OBJECTIVE-ATTACK-POSTURE-ATTACK-THRESHOLD ARMIES CAADEMO MEMBER))
   (REGION (721 101 116 59)
           NIL
           NIL
           NIL
           ((ICON.REGION (654 102 437 48)) (EXPAND.REGION (721 101 116 59))))
   (SAVED.VALUES 5)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
   (TITLE "Attack-Attack")
   (TITLEFONT FONTS:HL108)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 116)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))

(DIGIACTUATOR01742
  ("AUGUSTE" "24-Nov-85 0:24:52")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 68)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED
    #Slot (BEHIND-OBJECTIVE-ATTACK-POSTURE-DEFEND-THRESHOLD ARMIES CAADEMO MEMBER))
   (REGION (721 33 115 68)
           NIL
           NIL
           NIL
           ((ICON.REGION (654 44 437 48)) (EXPAND.REGION (721 33 115 68))))
   (SAVED.VALUES 2)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
   (TITLE "Attack-Defend")
   (TITLEFONT FONTS:HL108)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 115)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))

(DIGIACTUATOR01743
  ("AUGUSTE" "24-Nov-85 0:25:16")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 61)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED
      #Slot (BEHIND-OBJECTIVE-DEFEND-POSTURE-ATTACK-THRESHOLD ARMIES CAADEMO MEMBER))
     (RANGE.INCR 0.1)
     (REGION (836 99 116 61)
             NIL
             NIL
             NIL
             ((ICON.REGION (654 102 437 48)) (EXPAND.REGION (836 99 116 61))))
     (SAVED.VALUES 5)
     (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
     (TITLE "Defend-Attack")
     (TITLEFONT FONTS:HL108)
     (TOPUNIT (NIL))
     (VALUE.WAS.SAVED T)
     (WIDTH 116)
     (WINDOW "A Flavor Instance"
             NIL
             NIL
             NIL
             ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


 (DIGIACTUATOR01744
   ("AUGUSTE" "24-Nov-85 0:25:38")
   NIL
   ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
   NIL
   ()
   ((BORDER 4)
     (FONT FONTS:METSI)
     (HEIGHT 67)
     (IMAGE.WAS.PAINTED NIL)
     (OBJECT.DISPLAYED
      #Slot (BEHIND-OBJECTIVE-DEFEND-POSTURE-DEFEND-THRESHOLD ARMIES CAADEMO MEMBER))
     (REGION (837 32 115 67)
             NIL
             NIL
             NIL
             ((ICON.REGION (654 43 437 48)) (EXPAND.REGION (837 32 115 67))))
     (SAVED.VALUES 2)
     (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
     (TITLE "Defend-Defend")
     (TITLEFONT FONTS:HL108)
     (TOPUNIT (NIL))
     (VALUE.WAS.SAVED T)
     (WIDTH 115)
     (WINDOW "A Flavor Instance"
             NIL
             NIL
             NIL
             ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


 (DIGIACTUATOR01745
   ("AUGUSTE" "24-Nov-85 0:25:56")
   NIL
   ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
   NIL
   ()
   ((BORDER 4)
     (FONT FONTS:METSI)
     (HEIGHT 60)
     (IMAGE.WAS.PAINTED NIL)
     (OBJECT.DISPLAYED
      #Slot (BEHIND-OBJECTIVE-DELAY-POSTURE-ATTACK-THRESHOLD ARMIES CAADEMO MEMBER))
     (REGION (951 102 103 60)
             NIL
             NIL
             NIL
             ((ICON.REGION (661 103 430 48)) (EXPAND.REGION (951 102 103 60))))
     (SAVED.VALUES 4)
     (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
     (TITLE "Delay-Attack")
     (TITLEFONT FONTS:HL108)
     (TOPUNIT (NIL))
     (VALUE.WAS.SAVED T)
     (WIDTH 103)
 (WINDOW "A Flavor Instance"
         NIL
         NIL
         NIL
         ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR01746
 ("AUGUSTE" "24-Nov-85 0:26:13")
 NIL
 ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:METSI)
  (HEIGHT 70)
  (IMAGE.WAS.PAINTED NIL)
  (OBJECT.DISPLAYED
   #Slot (BEHIND-OBJECTIVE-DELAY-POSTURE-DEFEND-THRESHOLD ARMIES CAADEMO MEMBER))
  (REGION (953 32 104 70)
          NIL
          NIL
          NIL
          ((ICON.REGION (661 42 430 48)) (EXPAND.REGION (953 32 104 70))))
  (SAVED.VALUES 1)
  (SUPER.IMAGE #Unit (IMAGE.PANEL01740 CAADEMO))
  (TITLE "Delay-Defend")
  (TITLEFONT FONTS:HL10B)
  (TOPUNIT (NIL))
  (VALUE.WAS.SAVED T)
  (WIDTH 104)
  (WINDOW "A Flavor Instance"
          NIL
          NIL
          NIL
          ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR02289
 ("AUGUSTE" "24-Nov-85 2:06:29")
 NIL
 ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:METSI)
  (HEIGHT 77)
  (IMAGE.WAS.PAINTED NIL)
  (OBJECT.DISPLAYED #Slot (NUMBER-OF-RESERVE-CORPS NEW-CORPS CAADEMO OWN))
  (REGION (668 344 238 77)
          NIL
          NIL
          NIL
          ((ICON.REGION (636 484 289 48)) (EXPAND.REGION (668 344 238 77))))
  (SAVED.VALUES 3)
  (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
  (TITLE "Number of Reserve Corps")
  (TITLEFONT FONTS:HL10B)
  (TOPUNIT (NIL))
  (VALUE.WAS.SAVED T)
  (WIDTH 238)
  (WINDOW "A Flavor Instance"
          NIL
          NIL
          NIL
          ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR02290
 ("AUGUSTE" "24-Nov-85 2:06:52")
 NIL
 ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:METSI)
```

```
    (HEIGHT 58)
    (IMAGE.WAS.PAINTED NIL)
    (OBJECT.DISPLAYED #Slot (NUMBER-OF-COMBAT-READY-DIVISIONS NEW-CORPS CAADEMO OWN))
    (REGION (831 440 256 58)
            NIL
            NIL
            NIL
            ((ICON.REGION (622 402 352 48)) (EXPAND.REGION (831 440 256 58))))
    (SAVED.VALUES 4)
    (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
    (TITLE "Number of Combat-ready Divisions")
    (TITLEFONT FONTS:HL10B)
    (TOPUNIT (NIL))
    (VALUE.WAS.SAVED T)
    (WIDTH 256)
    (WINDOW "A Flavor Instance"
            NIL
            NIL
            NIL
            ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR02292
  ("AUGUSTE" "24-Nov-85 2:07:53")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 72)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (ECHELON-SPACES-TO-ARMY-OBJECTIVE NEW-CORPS CAADEMO OWN))
   (REGION (511 426 257 72)
           NIL
           NIL
           NIL
           ((ICON.REGION (507 640 355 48)) (EXPAND.REGION (511 426 257 72))))
   (SAVED.VALUES 1)
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "Echelon Spaces to Army Objective")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 257)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIACTUATOR03185
  ("AUGUSTE" "24-Nov-85 3:28:36")
  NIL
  ((DIGIACTUATOR ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 59)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS ARMIES CAADEMO MEMBER))
   (REGION (616 126 90 59)
           NIL
           NIL
           NIL
           ((ICON.REGION (628 129 341 48)) (EXPAND.REGION (616 126 90 59))))
   (SAVED.VALUES 2)
   (SUPER.IMAGE #Unit (IMAGE.PANEL01708 CAADEMO))
   (TITLE "Max Distance")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 90)
   (WINDOW "A Flavor Instance"
                NIL
                NIL
                NIL
                ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIMETER02285
  ("AUGUSTE" "24-Nov-85 2:03:54")
  NIL
  ((DIGIMETER ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 64)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (F-TO-E-FORCE-RATIO NEW-CORPS CAADEMO OWN))
   (REGION (746 637 168 64)
                NIL
                NIL
                NIL
                ((ICON.REGION (730 564 244 48)) (EXPAND.REGION (746 637 168 64))))
   (SAVED.VALUES 1.5)
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "F-to-E Force Ratio")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 168)
   (WINDOW "A Flavor Instance"
                NIL
                NIL
                NIL
                ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIGIMETER02287
  ("AUGUSTE" "24-Nov-85 2:05:13")
  NIL
  ((DIGIMETER ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:METSI)
   (HEIGHT 72)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (WOULD-BE-F-TO-E-FORCE-RATIO NEW-CORPS CAADEMO OWN))
   (REGION (640 510 266 72)
                NIL
                NIL
                NIL
                ((ICON.REGION (637 545 315 48)) (EXPAND.REGION (640 510 266 72))))
   (SAVED.VALUES 5)
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "Would-be F-to-E Force Ratio")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 266)
   (WINDOW "A Flavor Instance"
                NIL
                NIL
                NIL
                ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(DIVISIONS
  ("" "23-Nov-85 18:02:09")
  (MILITARY-UNITS)
  ((CLASSES GENERICUNITS))
  NIL
  ()
  ())


(IMAGE.PANEL01708
  ("AUGUSTE" "24-Nov-85 0:05:35")
  NIL
  ((IMAGE.PANEL ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 12)
   (HEIGHT 296)
   (IMAGE.WAS.PAINTED (NIL))
   (IMAGES (#Unit (DIGIACTUATOR03185 CAADEMO) #Unit (IMAGE.PANEL01740 CAADEMO)
                                              #Unit (IMAGE.PANEL01709 CAADEMO)
                                              #Unit (VERTICAL.TRAFFIC.LIGHT01625 CAADEMO)))
   (OBJECT.DISPLAYED #KB (CAADEMO))
   (REGION (603 5 480 296)
           NIL
           NIL
           NIL
           ((ICON.REGION (537 570 269 48)) (EXPAND.REGION (603 5 480 296))))
   (TITLE "Armies ActiveImage Panel")
   (TITLEFONT FONTS:HL108)
   (TOPUNIT #Unit (ARMIES CAADEMO))
   (WIDTH 480)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW T) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (ICON))))
   (\IMAGE.PANEL (#Unit (IMAGE.PANEL01740 CAADEMO)))))


(IMAGE.PANEL01709
  ("AUGUSTE" "24-Nov-85 0:07:34")
  NIL
  ((IMAGE.PANEL ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 12)
   (HEIGHT 85)
   (IMAGE.WAS.PAINTED (NIL))
   (IMAGES (#Unit (DIGIACTUATOR01679 CAADEMO) #Unit (DIGIACTUATOR01678 CAADEMO)
                                              #Unit (DIGIACTUATOR01651 CAADEMO)))
   (OBJECT.DISPLAYED #KB (CAADEMO))
   (REGION (777 189 286 85)
           NIL
           NIL
           NIL
           ((ICON.REGION (751 217 269 48)) (EXPAND.REGION (777 189 286 85))))
   (SUPER.IMAGE #Unit (IMAGE.PANEL01708 CAADEMO))
   (TITLE "At Objective Thresholds")
   (TITLEFONT FONTS:HL108)
   (TOPUNIT (ARMIES CAADEMO))
   (WIDTH 286)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(IMAGE.PANEL01740
  ("AUGUSTE" "24-Nov-85 0:22:36")
  NIL
  ((IMAGE.PANEL ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 12)
   (HEIGHT 164)
   (IMAGE.WAS.PAINTED (NIL))
   (IMAGES (#Unit (DIGIACTUATOR01746 CAADEMO) #Unit (DIGIACTUATOR01745 CAADEMO)
                                              #Unit (DIGIACTUATOR01744 CAADEMO)
                                              #Unit (DIGIACTUATOR01743 CAADEMO)
                                              #Unit (DIGIACTUATOR01742 CAADEMO)
                                              #Unit (DIGIACTUATOR01741 CAADEMO)))
   (OBJECT.DISPLAYED #KB (CAADEMO))
   (REGION (709 21 360 164)
           NIL
           NIL
```

```
            NIL
            ((ICON.REGION (746 43 345 48)) (EXPAND.REGION (709 21 360 164))))
    (SUPER.IMAGE #Unit (IMAGE.PANEL01708 CAADEMO))
    (TITLE "Behind Objective Posture Thresholds")
    (TITLEFONT FONTS:HL10B)
    (TOPUNIT #Unit (IMAGE.PANEL01708 CAADEMO))
    (WIDTH 360)
    (WINDOW "A Flavor Instance"
            NIL
            NIL
            NIL
            ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND)))))


(IMAGE.PANEL02283
  ("AUGUSTE" "24-Nov-85 2:02:10")
  NIL
  ((IMAGE.PANEL ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 12)
   (HEIGHT 416)
   (IMAGE.WAS.PAINTED (NIL))
   (IMAGES
    (#Unit (SIMPLE.VALUE.DISPLAY03313 CAADEMO) #Unit (SIMPLE.VALUE.DISPLAY02294 CAADEMO)
                                               #Unit (VERTICAL.TRAFFIC.LIGHT02293 CAADEMO)
                                               #Unit (DIGIACTUATOR02292 CAADEMO)
                                               #Unit (VERTICAL.TRAFFIC.LIGHT02291 CAADEMO)
                                               #Unit (DIGIACTUATOR02290 CAADEMO)
                                               #Unit (DIGIACTUATOR02289 CAADEMO)
                                               #Unit (VERTICAL.TRAFFIC.LIGHT02288 CAADEMO)
                                               #Unit (DIGIMETER02287 CAADEMO)
                                               #Unit (VERTICAL.TRAFFIC.LIGHT02286 CAADEMO)
                                               #Unit (DIGIMETER02285 CAADEMO)
                                               #Unit (VERTICAL.TRAFFIC.LIGHT02284 CAADEMO)))
   (OBJECT.DISPLAYED #KB (CAADEMO))
   (REGION (301 311 802 416)
            NIL
            NIL
            NIL
            ((ICON.REGION (506 624 299 48)) (EXPAND.REGION (301 311 802 416))))
   (TITLE "New-Corps ActiveImage Panel")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT #Unit (NEW-CORPS CAADEMO))
   (WIDTH 802)
   (WINDOW "A Flavor Instance"
            NIL
            NIL
            NIL
            ((ICON.WINDOW T) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (ICON)))))


(IMAGES
  ("AUGUSTE" "23-Nov-85 23:23:54")
  ((ENTITIES GENERICUNITS))
  ((CLASSES GENERICUNITS))
  "Dummy parent unit for gauges."
  ()
  ((DELETE DELETE.ALL.IMAGES METHOD METHOD)
   (DELETE.ALL.IMAGES DELETE.ALL.IMAGES METHOD METHOD)
   (DONT.RECREATE.IMAGES.AFTER.KBLOAD ASKUSER NIL (ONE.OF T NIL ASKUSER))
   (RECREATE.ALL.IMAGES RECREATE.ALL.IMAGES METHOD METHOD)
   (SAVE.ALL.IMAGES SAVE.ALL.IMAGES METHOD METHOD)
   (USER.DELETE.ALL.IMAGES USER.DELETE.ALL.IMAGES METHOD METHOD)
   (USER.RECREATE.ALL.IMAGES USER.RECREATE.ALL.IMAGES METHOD METHOD)))


 MILITARY-UNITS
   [illegible] "23-Nov-85 17:58:37")
   [illegible]ES GENERICUNITS))
   [illegible]ASSES GENERICUNITS))
  [illegible]
    [illegible]NS NIL NIL (MILITARY-UNITS) NIL NIL)
    [illegible]RE NIL NIL ((ONE.OF ATTACK DEFEND DELAY WITHDRAW)) NIL ((CARDINALITY.MAX (1)))))
```

```
(NEW-CORPS
  ("AUGUSTE" "23-Nov-85 20:37:29")
  NIL
  (CORPS)
  NIL
  ()
  ((ARMY (#Unit (ARMY-2 CAADEMO)))
   (ASSIGNMENT-STATUS (RESERVE))
   (COMMITTED-TO NIL)
   (ECHELON-SPACES-TO-ARMY-OBJECTIVE (1))
   (ENGAGEMENT-STATUS (ENGAGED))
   (F-TO-E-FORCE-RATIO (1.5))
   (NEIGHBOR NIL)
   (NUMBER-OF-COMBAT-READY-DIVISIONS (4))
   (NUMBER-OF-RESERVE-CORPS (3))
   (POSSIBLE-RESERVE-ASSIGNMENT (#Unit (CORPS-2-2 CAADEMO) #Unit (CORPS-2-4 CAADEMO)))
   (POSTURE (DELAY))
   (RESERVE-FOR (#Unit (CORPS-2-2 CAADEMO)))
   (SUPPORT-STATUS (OKAY))
   (WOULD-BE-F-TO-E-FORCE-RATIO (5))
   (WOULD-BE-POSTURE (ATTACK))
   (\IMAGE.PANEL (#Unit (IMAGE.PANEL02283 CAADEMO)))))


(SIMPLE.VALUE.DISPLAY02294
  ("AUGUSTE" "24-Nov-85 2:08:39")
  NIL
  ((SIMPLE.VALUE.DISPLAY ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:HL7)
   (HEIGHT 55)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (ARMY NEW-CORPS CAADEMO OWN))
   (REGION (326 524 126 55)
           NIL
           NIL
           NIL
           ((ICON.REGION (508 508 151 48)) (EXPAND.REGION (326 524 126 55))))
   (SAVED.VALUES #Unit (ARMY-2 CAADEMO))
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "Army")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 126)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(SIMPLE.VALUE.DISPLAY03313
  ("AUGUSTE" "24-Nov-85 7:08:33")
  NIL
  ((SIMPLE.VALUE.DISPLAY ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:HL7)
   (HEIGHT 58)
   (IMAGE.WAS.PAINTED NIL)
   (OBJECT.DISPLAYED #Slot (RESERVE-FOR NEW-CORPS CAADEMO OWN))
   (REGION (326 458 128 58)
           NIL
           NIL
           NIL
           ((ICON.REGION (326 458 197 48)) (EXPAND.REGION (326 458 128 58))))
   (SAVED.VALUES #Unit (CORPS-2-2 CAADEMO))
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "Reserve-For")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
```

```
      (WIDTH 128)
      (WINDOW "A Flavor Instance"
              NIL
              NIL
              NIL
              ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


   (VERTICAL.TRAFFIC.LIGHT01625
     ("AUGUSTE" "23-Nov-85 23:23:58")
     NIL
     ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
     NIL
     ()
     ((BORDER 4)
      (FONT FONTS:TR18B)
      (HEIGHT 89)
      (IMAGE.WAS.PAINTED NIL)
      (LAST.VALUE.DISPLAYED (DEFEND))
      (OBJECT.DISPLAYED #Slot (POSTURE ARMIES CAADEMO MEMBER))
      (REGION (643 186 130 89)
              NIL
              NIL
              NIL
              ((ICON.REGION (627 213 138 48)) (EXPAND.REGION (643 186 130 89))))
      (RELAY.REGIONS.ALIST (((ATTACK 0 0 120 22) (DEFEND 0 22 120 22) (DELAY 0 44 120 22))))
      (SAVED.VALUES DEFEND)
      (SUPER.IMAGE #Unit (IMAGE.PANEL01708 CAADEMO))
      (TITLE "ARMIES's POSTURE")
      (TITLEFONT FONTS:HL10B)
      (TOPUNIT (NIL))
      (VALUE.WAS.SAVED T)
      (WIDTH 130)
      (WINDOW "A Flavor Instance"
              NIL
              NIL
              NIL
              ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


   (VERTICAL.TRAFFIC.LIGHT02284
     ("AUGUSTE" "24-Nov-85 2:03:17")
     NIL
     ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
     NIL
     ()
     ((BORDER 4)
      (FONT FONTS:TR18B)
      (HEIGHT 95)
      (IMAGE.WAS.PAINTED NIL)
      (LAST.VALUE.DISPLAYED (OKAY))
      (OBJECT.DISPLAYED #Slot (SUPPORT-STATUS NEW-CORPS CAADEMO OWN))
      (REGION (921 607 166 95)
              NIL
              NIL
              NIL
              ((ICON.REGION (755 609 219 48)) (EXPAND.REGION (921 607 166 95))))
      (RELAY.REGIONS.ALIST (((OKAY 0 0 156 24) (POOR 0 24 156 24) (DESPERATE 0 48 156 24))))
      (SAVED.VALUES OKAY)
      (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
      (TITLE "Support Status")
      (TITLEFONT FONTS:HL10B)
      (TOPUNIT (NIL))
      (VALUE.WAS.SAVED T)
      (WIDTH 166)
      (WINDOW "A Flavor Instance"
              NIL
              NIL
              NIL
              ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


   (VERTICAL.TRAFFIC.LIGHT02286
     ("AUGUSTE" "24-Nov-85 2:04:56")
     NIL
     ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
```

```
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:TR12B)
  (HEIGHT 96)
  (IMAGE.WAS.PAINTED NIL)
  (LAST.VALUE.DISPLAYED (ATTACK))
  (OBJECT.DISPLAYED #Slot (WOULD-BE-POSTURE NEW-CORPS CAADEMO OWN))
  (REGION (510 594 153 96)
          NIL
          NIL
          NIL
          ((ICON.REGION (637 610 239 48)) (EXPAND.REGION (510 594 153 96))))
  (RELAY.REGIONS.ALIST (((ATTACK 0 0 143 18) (DEFEND 0 18 143 18)
                                              (DELAY 0 36 143 18)
                                              (WITHDRAW 0 54 143 18))
                        ))
  (SAVED.VALUES ATTACK)
  (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
  (TITLE "Would-be Posture")
  (TITLEFONT FONTS:HL10B)
  (TOPUNIT (NIL))
  (VALUE.WAS.SAVED T)
  (WIDTH 153)
  (WINDOW "A Flavor Instance"
          NIL
          NIL
          NIL
          ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(VERTICAL.TRAFFIC.LIGHT02288
 ("AUGUSTE" "24-Nov-85 2:05:49")
 NIL
 ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:HL12B)
  (HEIGHT 89)
  (IMAGE.WAS.PAINTED NIL)
  (LAST.VALUE.DISPLAYED (DELAY))
  (OBJECT.DISPLAYED #Slot (POSTURE NEW-CORPS CAADEMO OWN))
  (REGION (921 509 166 89)
          NIL
          NIL
          NIL
          ((ICON.REGION (791 455 168 48)) (EXPAND.REGION (921 509 166 89))))
  (RELAY.REGIONS.ALIST (((ATTACK 0 0 156 16) (DEFEND 0 16 156 16)
                                              (DELAY 0 32 156 16)
                                              (WITHDRAW 0 48 156 16))
                        ))
  (SAVED.VALUES DELAY)
  (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
  (TITLE "Posture")
  (TITLEFONT FONTS:HL10B)
  (TOPUNIT (NIL))
  (VALUE.WAS.SAVED T)
  (WIDTH 166)
  (WINDOW "A Flavor Instance"
          NIL
          NIL
          NIL
          ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(VERTICAL.TRAFFIC.LIGHT02291
 ("AUGUSTE" "24-Nov-85 2:07:28")
 NIL
 ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
 NIL
 ()
 ((BORDER 4)
  (FONT FONTS:BIGFNT)
  (HEIGHT 79)
  (IMAGE.WAS.PAINTED NIL)
```

```
    (LAST.VALUE.DISPLAYED (ENGAGED))
    (OBJECT.DISPLAYED #Slot (ENGAGEMENT-STATUS NEW-CORPS CAADEMO OWN))
    (REGION (931 345 156 79)
            NIL
            NIL
            NIL
            ((ICON.REGION (636 404 248 48)) (EXPAND.REGION (931 345 156 79))))
    (RELAY.REGIONS.ALIST (((ENGAGED 0 0 146 28) (NOT-ENGAGED 0 28 146 28))))
    (SAVED.VALUES ENGAGED)
    (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
    (TITLE "Engagement Status")
    (TITLEFONT FONTS:HL10B)
    (TOPUNIT (NIL))
    (VALUE.WAS.SAVED T)
    (WIDTH 156)
    (WINDOW "A Flavor Instance"
            NIL
            NIL
            NIL
            ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


(VERTICAL.TRAFFIC.LIGHT02293
  ("AUGUSTE" "24-Nov-85 2:08:16")
  NIL
  ((VERTICAL.TRAFFIC.LIGHT ACTIVEIMAGES) IMAGES)
  NIL
  ()
  ((BORDER 4)
   (FONT FONTS:TR18B)
   (HEIGHT 73)
   (IMAGE.WAS.PAINTED NIL)
   (LAST.VALUE.DISPLAYED (RESERVE))
   (OBJECT.DISPLAYED #Slot (ASSIGNMENT-STATUS NEW-CORPS CAADEMO OWN))
   (REGION (516 347 130 73)
           NIL
           NIL
           NIL
           ((ICON.REGION (508 565 243 48)) (EXPAND.REGION (516 347 130 73))))
   (RELAY.REGIONS.ALIST (((RESERVE 0 0 120 25) (ONLINE 0 25 120 25))))
   (SAVED.VALUES RESERVE)
   (SUPER.IMAGE #Unit (IMAGE.PANEL02283 CAADEMO))
   (TITLE "Assignment Status")
   (TITLEFONT FONTS:HL10B)
   (TOPUNIT (NIL))
   (VALUE.WAS.SAVED T)
   (WIDTH 130)
   (WINDOW "A Flavor Instance"
           NIL
           NIL
           NIL
           ((ICON.WINDOW NIL) (EXPAND.WINDOW NIL) (CURRENT.WINDOW (EXPAND))))))


KBEnd
```

```
;;;  -*- Mode:LISP; Package:KEE; Base:10. -*-

(DEFUN |CAADEMO>CORPS:ASSIGN-CORPS!method| (SELF &AUX RESERVE-FOR)
    (PUT.VALUE 'CORPS 'CORPS-TO-BE-ASSIGNED SELF)
    (FORWARD.CHAIN 'ASSIGNMENT-OF-NEW-CORPS-RULES)
    (PUT.VALUE SELF 'ASSIGNMENT-STATUS 'RESERVE)
    (SETQ RESERVE-FOR (GET.VALUE SELF 'POSSIBLE-RESERVE-ASSIGNMENT))
    (PUT.VALUE SELF 'RESERVE-FOR RESERVE-FOR)
    (PUT.VALUE SELF 'ARMY (GET.VALUE RESERVE-FOR 'ARMY)))

(DEFUN |CAADEMO>CORPS::ASSIGN-CORPS-POSTURES!method| (SELF &AUX
                                                (CORPS (UNIT.ALLCHILDREN SELF 'MEMBER)))
    (LOOP FOR CORP IN CORPS DO (UNITMSG CORP 'ASSIGN-CORPS-POSTURE)))

(DEFUN |CAADEMO>CORPS:ASSIGN-CORPS-POSTURE!method| (SELF)
    (REMOVE.ALL.LOCAL.VALUES SELF 'POSTURE)
    (QUERY '(THE POSTURE OF ,SELF IS ?POSTURE)
           1
           'ASSIGN-CORPS-POSTURE-RULES))

(DEFUN |CAADEMO>CORPS:COMMIT-RESERVE-CORPS!method| (SELF)
    (PROG
     ((ASSIGNMENT-STATUS (GET.VALUE SELF 'ASSIGNMENT-STATUS))
      (SUPPORT-STATUS (GET.VALUE SELF 'SUPPORT-STATUS))
      (NUMBER-OF-COMBAT-READY-DIVISIONS (GET.VALUE SELF 'NUMBER-OF-COMBAT-READY-DIVISIONS))
      (MIN-NUMBER-OF-COMBAT-READY-DIVISIONS (GET.VALUE SELF
                                                 'MIN-NUMBER-OF-COMBAT-READY-DIVISIONS))
      (ARMY (GET.VALUE SELF 'ARMY))
      MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS
      ONLINE-CORPS
      OC-SPACES-TO-ARMY-OBJECTIVE
      ONLINE-CORPS-POSTURE
      NEIGHBORS
      COMMITTED-TO)
     (REMOVE.ALL.LOCAL.VALUES SELF 'COMMITTED-TO)
     (COND ((NOT (EQUAL ASSIGNMENT-STATUS 'RESERVE))
            (PRINTOUT T T "This is not a reserve unit !!")
            (RETURN 'FAIL-1))
           ((NOT (EQUAL SUPPORT-STATUS 'OKAY))
            (PRINTOUT T T "The support status of this unit is not 'okay !!")
            (RETURN 'FAIL-2))
           ((< NUMBER-OF-COMBAT-READY-DIVISIONS MIN-NUMBER-OF-COMBAT-READY-DIVISIONS)
            (PRINTOUT T T "There are not enough combat ready divisions in this corps !!")
            (RETURN 'FAIL-3)))
     (SETQ MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS (GET.VALUE ARMY
                                                 'MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS))
     (SETQ ONLINE-CORPS (GET.VALUE SELF 'RESERVE-FOR))
     (SETQ OC-SPACES-TO-ARMY-OBJECTIVE (GET.VALUE ONLINE-CORPS
                                              'ECHELON-SPACES-TO-ARMY-OBJECTIVE))
     (SETQ ONLINE-CORPS-POSTURE (GET.VALUE ONLINE-CORPS 'POSTURE))
     (SETQ NEIGHBORS (GET.VALUES ONLINE-CORPS 'NEIGHBOR))
     (COND ((AND (> OC-SPACES-TO-ARMY-OBJECTIVE 0)
                 (NOT (EQUAL 'ATTACK ONLINE-CORPS-POSTURE))
                 (LOOP FOR
                       NEIGHBOR
                       IN
                       NEIGHBORS
                       WHEN
                       (> (- OC-SPACES-TO-ARMY-OBJECTIVE
                             (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE))
                          MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS)
                       COLLECT
                       NEIGHBOR))
            (PUT.VALUE SELF 'COMMITTED-TO ONLINE-CORPS)
            (PRINTOUT T T "Committed to " (UNIT.NAME ONLINE-CORPS))
            (RETURN ONLINE-CORPS)))
     (COND
      ((SETQ
        COMMITTED-TO
        (CAR (LOOP FOR
                   NEIGHBOR
                   IN
                   NEIGHBORS
                   WHEN
                   (AND (> (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE) 0)
                        (NOT (EQUAL 'ATTACK (GET.VALUE NEIGHBOR 'POSTURE)))
                        (> (- (GET.VALUE NEIGHBOR 'ECHELON-SPACES-TO-ARMY-OBJECTIVE)
                              OC-SPACES-TO-ARMY-OBJECTIVE)
                           MAXIMUM-DISTANCE-BETWEEN-NEIGHBORS))
                 COLLECT
                 NEIGHBOR)))
   (PUT.VALUE SELF 'COMMITTED-TO COMMITTED-TO)
   (PRINTOUT T T "Committed to " (UNIT.NAME COMMITTED-TO))
   (RETURN COMMITTED-TO)))
 (PRINTOUT T T "Guess nobody needs my help !!")
 (RETURN 'FAIL-5)))
```

APPENDIX I

DISTRIBUTION

Addressee	No of copies
Deputy Chief of Staff for Research, Development, and Acquisition Headquarters, Department of the Army ATTN: DAMA-ZA Washington, DC 20310	1
Deputy Under Secretary of the Army (Operations Research) Washington, DC 20310	1
Director US Army TRADOC Analysis Center White Sands Missile Range, NM 88002	2
Director US Army Materiel Systems Analysis Agency Aberdeen Proving Ground, MD 21005	1
Director US Army Ballistics Research Laboratories Building 305 Aberdeen Proving Ground, MD 21005	1
Commander TRADOC Analysis Command ATTN: ATOR-CA BGEN John D. Robinson Fort Leavenworth, KS 66027	1
Commander Foreign Science and Technology Center 227th Street NE Charlottesville, VA 22901	1

Addressee	No of copies
Commander Army Research Institute 5001 Eisenhower Avenue Alexandria, VA 22333	2
Defense Technical Information Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22314-6145	2
The Pentagon Library (Army Studies Section) ATTN: ANRAL-RS The Pentagon Washington, DC 20310	1
Commandant US Army War College ATTN: Library Carlisle Barracks, PA 17013	1
Commandant US Army War College ATTN: CLW Carlisle Barracks, PA 17013	1
Air University ATTN: AU CADRE/WGTA (Capt. Taylor) Maxwell Air Force Base, AL 36112-5000	1
President National Defense University ATTN: NDU-LD-CDC Washington, DC 20319-6000	1
Superintendent United States Military Academy ATTN: Mathematics Department West Point, NY 10996	1

Addressee	No of copies
Superintendent United States Military Academy ATTN: Engineering Department West Point, NY 10996	1
Naval Postgraduate School ATTN: Department of Operations Research Code 55PY (Professor Parry) Monterey, CA 93940	1
Commandant US Army Infantry School ATTN: ATSH-IVT Fort Benning, GA 31905	1
Commander in Chief United States Readiness Command ATTN: RCDA MacDill Air Force Base, FL 33608	1
CINCPAC Staff C3S ASII Box 32A Camp Smith, HI 96861	1
Commander US Army Western Command ATTN: APOP-SPM Fort Shafter, HI 96858-5100	1
Commander US Army, Japan ATTN: AJCS APO San Francisco 96343	1
Commander/Director US Army Engineer Studies Center Casey Building, No. 2594 Fort Belvoir, VA 22060	1

Addressee	No of copies
Commander US Army Ballistic Missile Defense Systems Command Huntsville, AL 35807	1
Commander US Army Training and Doctrine Command Fort Monroe, VA 23651	1
Commander US Army Materiel Command 5001 Eisenhower Avenue Alexandria, VA 22333	1
Air Force Center for Studies and Analyses AFCSA/SAMI Room 1D363, Pentagon Washington, DC 20330-5425	1
Headquarters Foreign Technology Division (AFSC) Wright-Patterson Air Force Base, OH 45433	1
President Center for Naval Analyses 4401 Ford Avenue Post Office Box 16268 Alexandria, VA 22302-0268	1
Naval Research Laboratory ATTN: Code 5704 4555 Overlook Avenue Washington, DC 20375	1
CINC, US REDCOM ATTN: RCSTA MacDill Air Force Base, FL 33608-6001	1

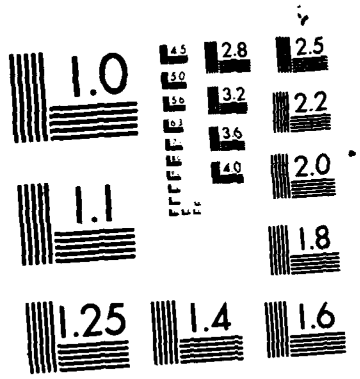
Addressee	No of copies
Commander Army Materiel Command AMC-DMA-MS (Dr. John Lazaruk) 5001 Eisenhower Avenue Alexandria, VA 22333	1
Army Artificial Intelligence Center Headquarters, Department of the Army DACS-DMA (ATTN: LTC David Tye) Washington, DC 20310-0700	1
Mr. Wendell G. Sykes Arthur D. Little, Inc. Acorn Park Cambridge, MA 02140-2390	1
OJCS/J8 ATTN: Mr. Vince P. Roske Pentagon Room 1D928A Washington, DC 20313	1
MITRE Corporation C^3I Division: Systems Engineering Analysis ATTN: Mr. Richard D. Nugent 1820 Dolley Madison Blvd McLean, VA 22102	1
The RAND Corporation ATTN: Mr. Carl Builder 1700 Main Street Santa Monica, CA 90406-2138	1
OJCS ATTN: Dr. John Dockery Pentagon Room 1B737 Washington, DC 20310	1

Addressee	No of copies
Defense Advanced Research Projects Agency ATTN: Mr. Stephen H. Kaisler 1400 Wilson Blvd Arlington, VA 22209-2308	1
US Army Research Institute Computer Center 5001 Eisenhower Avenue Alexandria, VA 22333	1
Organization of the Joint Chiefs of Staff (J-4) Studies, Concepts, and Analysis Division ATTN: MAJ Donald E. Fowler The Pentagon Washington, DC 20310-5000	1
Commander US Army Intelligence Center US Army Information Systems Command ATTN: ATSI-SA (MAJ Brand) Ft. Huachuca, AZ 58613	1
Commander US Army Logistics Center ATTN: ATCL-OPT (Mr. Oliver Hedgepeth) Building 12401, Room 245 Ft. Lee, VA 23801-6000	1
Commander US Army Combat Developments Experimentation Center Directorate of Information Management ATTN: Mr. Russ Davis Ft. Ord, CA 93941	1
Commander US Army Signal School ATTN: ATZH-PO (CPT Schoedel) Ft. Gordon. GA 30905	1

Addressee	No of copies
Commander US Army Signal School and Fort Gordon ATTN: ATZH-CLC-A (CPT Routh) Ft. Gordon, GA 30905	1
Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211	1
United States Military Academy Office of AI Analysis and Evaluation ATTN: AMDN-B (MAJ Judy) West Point, NY 10996-1695	1
US Army ITAC Bldg 203 Stop 314 ATTN: AIAT-N (Mr. Hubbard) Washington Navy Yard Washington, DC 20374-2136	1
Office of the Secretary of Defense Directorate of Computer Support ATTN: Mr. Joseph M. Powers Room 1C 730, Pentagon Washington, DC 20301	1
US Army TRADOC Analysis Center ATTN: Mr. Warren Olson White Sands Missile Range, NM 38002	1
Director US Army Models Management Office Fort Leavenworth, KS 66027	1

AD-A181 029 ARTIFICIAL INTELLIGENCE STUDY (AIS)(U) ARMY CONCEPTS ANALYSIS AGENCY BETHESDA MD R B MODJESKI FEB 87 CAA-RP-87-1 3/3

UNCLASSIFIED F/G 15/6 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

Addressee	No of copies
Internal Distribution	
Director	1
Assistant Director, RS	1
Assistant Director, RQ	1
Assistant Director, SP	1
Assistant Director, FO	1
Assistant Director, FS	1
CAA Library	2
Chief, Advanced Research Projects Office	1

GLOSSARY

1. ABBREVIATIONS, ACRONYMS, AND SHORT TERMS

AI	artificial intelligence
ALBM	Air Land Battle Management Model
ALU	Arithmetic Logic Unit
AMIP	Army Model Improvement Program
CAA	US Army Concepts Analysis Agency
C^2	command and control
C^3	command, control, and communications
C^3I	command, control, communications, and intelligence
DARPA	Defense Advanced Research Projects Agency
ISD	Instructional Systems Development Model
MIMD	multiple instruction multiple data computer hardware
MIPS	millions of instructions per second
SIMD	single instruction multiple data computer hardware
VHSIC	very high speed integrated circuit
VLSI	very large-scale integration

2. DEFINITIONS

active values
A knowledge representation methodology which allows values in the knowledge base to be tied to graphic icons. These graphic images allow the user to change the value of the knowledge base by changing the icon on the computer screen display Active values are frequently used to drive graphics in AI programs in the form of gauges which show the value of the program variable.

algorithm
A step-by-step list of instruction for solving a particular problem. A well-defining beginning and end point is assumed. The scope of the problem is also assumed to be finite and amenable to mathematical optimization techniques.

applicative programing
A style of problem solving and programing characterized by LISP.

architecture
The organization of a large system of hardware, software, or both. In AI software, the term refers to the application program or the language used to develop the application.

artificial intelligence
A subarea of computer science which studies symbolic (i.e., nonnumerical) processes in computers. AI uses programing techniques to represent knowledge that is used by the computer to make an inference. The computer, its programs, and the user achieve a limited type of intelligence behavior in a narrow domain (e.g., decisionmaking, problem solving). The underlying technical issues in developing AI systems are representing knowledge, controlling search, and inference.

Audit trail (Software)
A systematic documentation of software code and logic that support decisions about implementation of a computer program. Maintenance of a complete audit trail enables software modifications in a logical, coordinated, and systematic manner.

automatic programing
An AI technique that allows programs to reason about themselves. Automatic programing helps programers manage large programs or can produce small programs from a natural (i.e., noncoded) description of what a program is to do. An automatic program can produce code and verify it in a high level language such as PL-1 or FORTRAN (called the target language) from a programer's English input of examples of the program's I/O pairs or output.

backward chaining
An inference method that starts with what it wants to prove and then establishes evidence of facts needed to support that inference (i.e., top-down processing). Backward chaining is used by programers as a control strategy to order chains of inference in a knowledge base. The chaining mechanism proves a tentative goal by backing up to the "if" part of a rule to determine its correctness. This leads to searching for other rules that confirm the "if" part of the rule. For example, to prove C, the program starts with the rule "If A and B, then C." A and B are proven by establishing additional facts (as determined by other rules). The process continues recursively, expanding backward until a solution is reached (see forward chaining).

bitblt
Bit boundary block transfer

bit-mapped displays
Computer screen displays where each pixel is tied to a memory cell.

blackboard
A modular system architecture (i.e., AI control strategy) that employs a global data base memory that can be called by several processes called knowledge sources. The memory that is common to all processes serves as a basis for communication of intermediate results among rules or knowledge sources. The blackboard methodology provides a means for rules to be tied together to produce some tentative inferences. The blackboard is like the physical blackboard that crimesolvers use to list clues and tentative solutions. This design allows many separate knowledge bases to each examine a common working memory.

CDR coding
It is sometime possible to economize storage requirements or shrink the working-set size by changing the implementation strategy for data structures. The primary compound data structure (sic in LISP) is the CONS cell, which is simply a pair of pointers to other objects. Typically these CONS cells are used to represent lists and for that case, it has been observed that the CDR part of the CONS cell often happens to be allocated sequentially after the CONS. As a compaction scheme and as a strategy for increasing the locality (and hence reducing the working-set), a method called CDR-coding was developed that allows a CONS cell to efficiently state that the CDR is the next cell in memory. (Gabriel, 1985, p. 14)

class variable
A variable shared by an entire class of objects. Class variables share their values with all instances of a class of objects.

cognition (also Cognitive Process)
1. A generic term for any process whereby an organism becomes aware or obtains knowledge of an object. Although it is part of the traditional terminology and has subjective connotations, many neobehaviorists use the term. It includes perceiving, recognizing, conceiving, judging, reasoning. Some older authorities held that sensing was not strictly cognitive (holding that a distinct cognizing process followed upon sensing), but in modern usage sensing is usually included under cognition. In most systems, cognition, affection, and conation are the three categories under which all mental processes are classified. 2. The product cognizing; the knowledge obtained (not recommended). 3. The awareness of objects. This usage usually results from failure to distinguish (1) from (2). 4. (Behavioral) a hypothetical stimulus-stimulus association or perceptual organization inferred to account for expectancies. (English and English, 1958, pp. 92-93).

command and control
A system that connects a commander to many sensor, support, and effector elements in order to have all these elements work together in a unified, integrated, and controlled manner toward some otherwise unattainable goal. Human decisionmaking is assumed to take place within the structure of command and control.

COSAGE

Combat Sample Generator. A high-resolution, division-level simulation model which simulates a day's combat activity to generate ammunition consumption and equipment and personnel loss data. COSAGE is a two-sided, stochastic model written in the SIMSCRIPT II.5 programing language. The determining factor for the core requirements is the number of concurrent small unit battles which occur, the total number of units played (from platoon to division level), and the number of orders for these units. Orders are given for maneuver units, and if given at the battalion level, for example, will be executed by every maneuver platoon under that battalion. The gamer can direct a maneuver unit to do one of three things: (1) move; (2) attack; (3) defend. Indirect fire units act and react dynamically. In addition to combined arms ground combat, COSAGE can simulate attack helicopters, mines, visibility restrictions, weather conditions, smoke, illumination, tactical aircraft, and air defense.

data abstraction

A principle of object oriented programing. Data types are characterized by operation on their values. Programing the operations are methods of a class. This is representative of a data type, and the values are instances of the data type.

data sets

Unique combinations of aggregations of data elements. Examples are the sales order, accounts receivable ledger card, sales summary report, payroll register, etc. It should be noted that a data set is a potential combination of data elements. Not all data elements need be present at one time. For example, a payroll register entry for a given employee may contain only one or two of several possible deductions. (Sippl, 1985, p. 118)

data structure

A structure for organizing information within a computer and giving the data significance beyond the binary numeric value held by the bits which make up the structure. A character string for example, is a data structure and signifies that a set of bytes represents a string of ASCII characters rather than a sequence of integers. (Sippl, 1985, p. 119)

data type

The set of attributes that determine how the data in a given location is organized and treated. In low level programing--the machine and assembly levels--data has no type, and a programer must be aware of any details of any data structuring in order to use it. In higher level languages, data typing is enforced by both the compiler and by run time constraints. The internal structure of data objects may remain transparent to the programer. At the highest level, data may be entirely abstract, being restricted not only in terms of structure but by a range of applicable operations as well. (Sippl, 1985, p. 120)

demon (also deamon)
A LISP language control structure that suspends processes until an event occurs. The deamon is turned on, performs its job, and waits for the next event or termination. Demons are used to make inferences about new information, perform system housekeeping functions, or flag significant occurrences.

declarative knowledge
Facts representing knowledge. An AI knowledge representation methodology that tells a computer memory representing human knowledge what to know.

domain
A class of tasks within a field of knowledge. A domain narrows the scope of problem solving in an expert system.

dynamic scoping
Dynamic scoping means that any procedure can call any other, and variable values are passed down the control chain rather than being determined by the static block structure. Once a variable is declared or otherwise allocated during procedure A, it can be accessed from within any procedure B that A calls, or any procedure C that B calls, etc. This flow of variables occurs without reference to where procedures A, B, or C appear in the actual program text (Cohen and Feigenbaum, Vol II, p. 33).

expert system
A subfield of artificial intelligence programing. Software is developed that mimics the reasoning process of human experts in limited topics. Expert systems perform such tasks as problem solving in specialized areas of expertise called a domain. The domain knowledge is called a knowledge base. The knowledge of multiple human experts can be programed into the problem solving software. Expertise involves the representation of knowledge plus the domain-specific strategies for using the knowledge flexibly to produce a correct result (process control). Expert system programs have (1) a knowledge base to store facts and heuristics; (2) an inference engine to process the knowledge base and draw an inference; and (3) interfaces to explain how the inference was made and then acquire new knowledge.

Fifth Generation Project
A research project sponsored by the Government of Japan to develop advanced hardware and software for AI applications. The project currently is exploring AI applications on parallel processing computers.

flavors
An object-oriented extension of the LISP language which allows programer defined data types. Flavors allow the construction of structured abstract objects that bundle up procedural and declarative information about the thing represented by the object. Each flavor is a generic operator. A flavor contains data (numbers or symbols) and

operations on the data (methods). The objects are flavor instances that can be manipulated by sending messages, which are requests for specific operations. Flavors also allow property lists to become associated with objects (see object-oriented programing).

forward chaining
A problem solving methodology used in AI. It is characterized by working forward from known facts (rules) toward a conclusion. This inference methodology is used to control searching of the knowledge base. The software looks for all the "if" statements that are true in a global data base (i.e., knowledge base) of "if-then" rules. The "if" portion of the rules is compared to facts and new rules are added. The process continues until a goal is reached and no further inference can be made (see backward chaining).

frame
An object-oriented knowledge representation methodology. Defines a concept or object in terms of its attributes or properties (i.e., property lists). The property is called a slot. Each slot can have procedures attached to the slot that allow computation of its value. Self-contained pieces of code are attached to the slots. A frame is a data structure that includes declarative and procedural information in predefined internal relations (Cohen and Feigenbaum, 1981, Vol II, p 158). Frames serve to partition knowledge into discrete structures which are described by slots. Multiple frames are joined together in an inheritance hierarchy which transmits slot values among interrelated frames without multiple specification in the programing of each frame.

fuzzy reasoning
A multivalued, syllogistic reasoning scheme derived from Boolean set theory. Fuzzy logic allows the major and minor premise to contain probabilistic quantifiers (e.g., 80 percent, most, few) rather than the two valued quantifiers of all or some found in classic logic. The result is a syllogism that allows transitivity of set containment and is used to represent uncertain knowledge. Arbitrary predicates in this type of reasoning allow greater flexibility in programing.

garbage collection
The reclamation of unreferenced objects in the memory storage of a LISP machine.

graphic icons
A bit-mapped display of an image on a computer screen. This graphic image is a graphic representation of a LISP object which resides inside the machine. LISP objects may be viewed and modified with these graphic representation. Complex graphics may be easily incorporated in the LISP program with graphic icons.

heuristic
Rules of thumb that help experts use domain-specific knowledge.

IF-THEN rule
A structural statement of relationship between facts. IF-THEN rules may structure defining relationship (IF it moves toward you and fires on you and is not identified as Blue, THEN Red). IF-THEN rules may also structure heuristics (IF fired upon, THEN take cover).

inheritance
A knowledge representation methodology based on associative memory research in cognitive psychology. Structure of a data base (i.e., knowledge base) will create a semantic network (a model of human memory) from which the inference engine subroutine of an expert system program can make an inference. Inheritance is also called property inheritance. In programing, inheritance allows an object to be placed into a class matrix. Variables and methods are inherited from a larger class of objects. Any variable defined in the larger class will appear in objects that are instances of the larger class.

inference
The derivation of a proposition from other propositions. A complex series of inferences can be organized to proceed from antecedent propositions that are given to whatever consequent propositions are justified, in which case, the forward chaining process is called data-driven inference; or it can start from a specification of the desired consequences and proceed by training to prove antecedents that will justify concluding the consequent, in which case the backward chaining process is called model-directed or goal-directed inference (Brownston, Farrell, Kant, and Martin, 1985, p. 448). For example, if an artillery piece fires short of its target, then the barrel is cold can be combined with a rule of inference (i.e., if A implies B and we know A for a fact, then we can assume B (**modus ponens**)). We are now given the fact that rounds fired from artillery land short, and we can infer a new fact (i.e., the gun is cold). Software using inferences can use thousands of rules to infer a new fact.

inference engine
The portion of an expert system software that has the inference and control strategies. As an application program, it can perform inferences to yield new knowledge in the knowledge domain. In some software, the inference engine also contains utilities for user interface and explanations. Inference engines contain different control methodologies such as forward and/or backward chaining, blackboards, production systems, etc. Given many alternative problem solving steps, an inference engine will choose the next action.

instance variable
A slot (see slots)

instantiation
In object-oriented programing instantiation is allowing objects to take on specific properties (i.e., replacing variables with constants). A generic tank becomes an M1 with a serial number of 12286 and a crew of three. Instance variables can be called slots.

intelligence (human)

I. There is more agreement on the behaviors referred by the term than there is on how to interpret or categorize them. Three concepts recur frequently in attempts to state its connotations: that of ability to deal effectively with tasks involving abstractions; that of ability to learn; and that of ability to deal with new situations. Popular opinion assumes that "real intelligence" is innate, but this is rejected from professional use of the term. The first two of the following definitions limit themselves to stating operations by which intelligence is to be distinguished from other constructs; the third is widely accepted description. 1. That hypothetical construct which is measured by a properly standardized intelligence test. This definition sounds circular but is not: intelligence tests can be--and in fact have been--devised and standardized without having any particular or clear definition of intelligence. 2. (H. English) The individual's total repertory of those problem-solving and cognitive-discrimination responses that are usual and expected at a given level and in the large population unit to which he belongs. The usual and expected response has been defined by implication of test standardization as one of which 65 percent to 75 percent of the given population are capable. What is thus usual and expected changes qualitatively as well as quantitatively with age and with the population; intelligence tests, regarded as samplings of the total repertory, must reflect these changes. The intelligence level is measured by the proportion of the responses, usual and expected in the population, that an individual manifests in a standardized sample of task-demand situations. This definition leaves open the question of the organization of these responses--CP factor/general and factor theory. 3. (G. Stoddard) The ability to undertake activities that are characterized by difficulty, complexity, abstractness, economy, adaptiveness to a goal, social value, and the emergence of originals. 4. (popular) The rating a person obtains on an intelligence test; or, more loosely, the intelligence level. 5. (historical) Intellect. 6. (historical and popular) The capacity to profit by experience. (English and English, 1958, p 268). II. Through contact and interaction with the environment people acquire knowledge and skills which then manifest themselves in subsequent behavior. What a person knows and knows how to do at any point in time is a major dimension or property of behavior which enters importantly into descriptions and explanations of human activities. While there are many ways to elaborate and to categorize the different manifestations of this property it is known collectively as intelligence (i.e., intelligent behavior). (Bourne, Ekstrand, and Dominowski, 1971, p. 242)

intelligence (machine)

The developed capability of a device to perform functions that are normally associated with human intelligence, such as reasoning, learning, and self-improvement. (Related to machine learning.) Machine intelligence uses steps in developing real-world systems concerning identifying the problem and goals; acquiring the knowledge base; selecting a knowledge system structure and development tools; implementing the problem/inference subsystem; developing an intelligent interface; and testing, evaluating, and refining systems. Also Machine Learning: concerns the ability of a device to improve its performance based on its past performance. (Related to artificial intelligence.) (Sippl, 1985, pp. 230 and 266)

knowledge
1. The result of knowing. Simple knowledge is called apprehension (which includes perceiving); more complex is called comprehension or understanding (which includes awareness of relations, meaning, etc.). 2. The body of understood information possessed by an individual or by a culture. 3. That part of a person's information which is in accord with established fact. (English & English, 1958, p. 284)

knowledge base
A highly structured data base that interrelates all of its elements. Facts and heuristics about a domain are coded into the knowledge base. These facts and heuristics may take the form of rules, objects, logic, or working memory elements. Elements of the knowledge base may be used as data or programs in expert system software. The knowledge base may also be used to answer questions about the domain represented as well as providing access to the data.

knowledge representation
The methodology used to encode and store facts and relationships. It is a combination of data structures and interpretive procedures in a program that leads to knowledgeable behavior. List-and-pointer data structures can represent facts or rules in programing languages such as LISP. Knowledge is structured in the computer software by layering in rule memory, procedural rules, and functions and procedures (i.e., rules for conflict resolution). A domain (e.g., command and control problem solving based on sensor reports) is in the rule memory.

LIPS
Logical inference per second; a benchmark for measuring the speed of AI hardware.

LISP
List Processing computer programing language. LISP is a flexible, indefinitely extensible programing language that is used in many AI programing tasks. LISP is used for symbolic processing application programs.

logic
A formal system of rules used to manipulate symbols to yield an inference.

logic programing
A style of problem solving and programing characterized by the use of PROLOG.

MACSYMA
A symbolic computer program with procedures for helping people perform complex applied mathematics. MACSYMA can be considered an AI software toolkit.

message
A technique used in object-oriented programing to specify operations on an object. It is similar to a procedure call. Operations to be performed are named indirectly. Different objects can respond differently to the same message. Objects send other objects messages to cause a behavior to occur in a simulation.

metaknowledge
Interrelated multilevel knowledge.

methodology (analysis research)
Acquiring the necessary supporting and background technical information (forces, financial, logistical, concept, etc.) necessary for accomplishment of a given task, including model development when needed.

model development
The creation of a new computer system production support model or modification of an existing production support model. Model development involves a computer system program development effort and consideration of the research information and sponsor requirements.

model driven expert system
A type of expert system used for diagnostic purposes. It is usually driven by a backward chaining mechanism in its inference engine. Conceptually, it uses a model of the structure and behavior of the entity it is trying to understand.

model evaluation
The computer analysis of the production support model which provides the conceptual output for the sponsor. Model evaluation also required preparation of reports and studies to support the computer analysis.

object
The most primitive element used in object-oriented programing style. Objects are used to store program procedures and data. Objects store data in the form of variables. Objects respond to messages with behaviors (i.e., carrying out procedures).

object-oriented programing
A style of problem solving and programing characterized by the use of objects.

parallel processing
An advanced computer hardware design which allows the computation of multiple processes at the same time.

pattern-directed inference system

Any system composed of several Pattern-Directed Modules (PDMs), one or more data structures that may be examined and modified by the pattern-directed modules, and executive program to schedule and run the Pattern-Directed Modules is called a pattern-directed inference system (PDIS). In effect, a PDIS factors complex problems into manageable, largely independent subproblems. (Sippl, 1985, p. 344)

pattern-directed modules

The type of computer program that is used to develop an expert system cannot have its flow of control and data utilization rigidly fixed because such a structure is ill-adapted for simulating a human's responses to a complex, rapidly changing unfamiliar environment. Instead, such a program must examine the state of the world at each step of the decision process and react appropriately because new stimuli continually arise. The type of program that has been delivered to cope with this constant change is a loosely organized collection of PDMs that detect situations and respond to them. (Sippl, 1985, p. 344)

pattern matching

The process of comparing string patterns to locate matching or nearly matching strings. Pattern matching is used in text processing software and is especially important in artificial intelligence, where pattern matching and parallel processing are used to simulate thinking behavior of the human brain. (Sippl, 1985, p. 344) In expert system using production rules the matching operator searches for equivalence between left-hand sides of rules (i.e., the "IF" portions of a "IF-THEN" statement) within the data memory to find all the combinations where rules can be satisficed (i.e., meets the minimal criteria of acceptability to solve a problem) with consistent binding (i.e., instantiations). The pattern matching process provides pattern-directed inference systems with an organizational scheme to use pattern-directed modules (i.e., program modules) on data memory (i.e., global memory).

pattern systems

These are systems or antecedent-driven systems. A consequent-driven (backward-chaining) system uses rule consequents (which represent goals) to guide the search for rules to "fire." The system collects those rules that can satisfy the goal in question and tries to satisfy the consequents of those rules, which usually represent the values of variables. In order to find these values, the values of the rule antecedent must be found. To satisfy each antecedent which represents a subgoal, the system collects those rules whose consequents satisfy its value. (Sippl, 1985, p. 344)

personnel training

Determining, arranging for, scheduling, and/or approval of education and training for assigned civilian and military personnel.

pixel

A term for picture element. Pixels are dots on a computer screen. In many AI machines one memory cell is tied to one pixel on the computer screen. Images called graphic icons are constructed by having the computer software write into the appropriate group of memory cells. This software output causes an image (also called bit-mapped display in many AI machines) to appear on the computer screen. In many AI graphics systems the bit-mapped pixel images are data structures.

procedural knowledge

An AI knowledge representation methodology that tells a computer memory what to do.

procedural programing

A problem solving and programing method which uses nested subroutines to organize and control program execution. PASCAL and APL are examples or procedural programing languages.

production system

A system containing IF-THEN statements (i.e., production rules) with conditions that may be satisfied in a data base and actions that may change the data base. A control mechanism selects usable production rules in an effort to reach a goal state (problem solution). The production rules are stored in a global data base (data memory) and are used by an inference engine to control the selection of the rules. This style of problem solving and program uses IF-THEN rules to represent symbolic notions that characterize situations and their related actions (i.e., IF (condition), THEN (action)). IF conditions may be satisfied in a data base and THEN actions may change that data base.

PROLOG

PROgraming language for LOGic. An AI programing language based on predicate calculus. The Japanese have chosen PROLOG for their Fifth Generation AI Project.

property list

Since its inception, LISP has associated with each symbol a kind of tabular data structure called a property list (plist for short). A property list contains zero or more entries; each entry associates with a key (called the indicator), which is typically a symbol, an arbitrary LISP object (called the value or, sometimes, the property). There are no duplications among the indicators; a property list may only have one property at a time with a given name. In this way, given a symbol and an indicator (another symbol), an associated value can be retrieved. A property list is very similar in purpose to an association list. The difference is that a property list is an object with a unique identity; the operations for adding and removing property list entries are destructive operations that alter the property list rather than making a new one. Association lists on the other hand, are normally augmented nondestructively (without side effects) by adding new entries to the front...A property list is implemented as a memory cell containing a list with an even number (possible zero) of elements. (Usually this memory cell is the property list cell of a symbol, but any memory cell acceptable to setf can be used if getf and remf are used). Each pair of elements in the list constitutes an entry; the first item is the indicator, and the second is the value. Because property list functions are given the symbol and not the list itself, modifications to the property list can be recorded by sorting back into the property-list cell of the symbol. (Steele, et al., 1984, pp. 163-164)

Recursive-decent compiler
A local optimization for stack-based LISP machines. The compiler reads through the execution-order treewalk of a program, examines LISP expressions and determines the proper runtime code to execute. Gabriel (1985) has suggested that the speed at which a LISP compiler optimizes code is a separate issue from the speed: Simple compilers essentially eliminate the dispatching routine and generate calls to the correct routines, with some bookkeeping for values in between. Fancier compilers do a lot of open-coding, generating the body of a routine in-line with the code that calls the body of a routine in-line with the code that calls it...further optimization involves delaying the boxing or number CONSing of numbers in a numeric computation. Some compilers also rearrange the order of evaluation, do constant-folding, loop-unwinding, common-subexpression elimination, register optimization, cross optimizations (between functions), peephole optimization, and many of the other classical compiler techniques...when evaluating a compiler, it is important to know what the compiler in question can do. Often looking at some sample code produced by the compiler for an interesting piece of code is worthwhile evaluation technique for an expert. Knowing what is open-coded, what constructs are optimized, and how to declare facts to the complier in order to help it produce code are the most important things for a user to know. A separate issue is "How fast is the compiler?" In some case the compiler is slow even if the code it generates is fast; for instance, it can spend a lot of time doing optimization. (Gabriel, 1985, pp. 22-23)

recursion
1. A property of computer software that allows procedures to call themselves. Recursion is like a FORTRAN loop. 2. Recursion allows you to return to a statement, function, or process again and again. 3. If you do not understand the concept of recursion, GO TO 1.

representation
The way in which a system stores knowledge about a domain. Knowledge consists of facts and the relationship between facts.

rule (IF-THEN rule)
A conditional statement of two parts. The first part (the IF clause) establishes necessary conditions that must be met if a second part (the THEN or multiple (THENs) is to be acted upon. Given evidence, the inference engine of an expert system software system builds on the rule base and reasons through it in order to reach a conclusion. Rules are the data of an expert system that are manipulated by an inference engine in order to yield an inference.

rule based expert system
An expert system which uses a knowledge base programed as production rules.

satisficing
A problem solving methodology where the minimal criteria provides a cutoff value for a set of solutions, each of which is equally acceptable. Satisficing provides a necessary and sufficient condition for problem resolution. It is more cost effective than optimization in most problems.

schema
A methodology for representation of a single concept. Schema allows properties to be used as the defining attributes of a concept. Schemas are used in frame or procedural methodologies for knowledge representation in AI software. A production rule methodology of knowledge representation can also use schema by attaching them to slots without properties.

search
The process in running AI software where the knowledge base is searched systematically for a solution path. The program starts with an initial state and works its way toward a satisficing solution called a goal state.

semantic
Dealing with meaning in AI software.

semantic network
A knowledge representation methodology used in AI programing. The methodology is used to create a network of nodes. Each node can represent an object and its values. The nodes are linked in the network by their interrelationship (i.e., function).

Semantic Node
An object in a semantic network used in knowledge representation.

Semantic Node Methodology (also Semantic Network)
A class of knowledge representation formalism that uses objects and values as nodes in a tree-like structure. The nodes are linked with arcs or links that indicate the relationships between the various nodes. For example to link a fact like all 155SP howitzers are cannons we create two nodes 155SP howitzer and cannon and connect them with a "isa" linkage. The knowledge representation that would result would be "A 155SP howitzer isa cannon." Very complicated taxonomies of knowledge with inheritance can be structured with semantic node methodology.

simulation
A technique in AI software that models human behavior in an attempt to achieve intelligent behavior from the computer.

slot

An attribute associated by a node in a frame methodology. Slots are used in knowledge representation for AI software. Slots contain descriptive attributes of objects. Facts typically known about generic objects can be placed in slots. Slots also serve as a knowledge hook for specifying relationship between defining attributes of an object. The slot can serve as the place where knowledge fits within the frame. For example, the generic object "cat" may have slots for its name and breed, and a procedural slot for findings its mother and father. Slots may also contain pointers to other frames. Slots are used for variables that are stored locally (see class variable).

stack-oriented hardware

A stack is a list such that additions and deletions can only be made at the front of the list. As objects are "pushed" onto the stack, the stack pointer is incremented. As objects are "popped" from the stack the stack pointer is decremented. In many LISP machines special high-speed memories are served as stack buffers, and stacks are manipulated by stack pointers in hardware. In LISP machines stacks are used in three kind of operations that are needed throughout the LISP implementation: 1. evaluation of nested expressions (outer expressions are pushed onto the stack until the inner expression is evaluated; 2. interrupts (the current environment is pushed onto the stack until the interrupt service routine is done); 3. function calls (function arguments and temporary local storage are saved on the stack). Since every LISP expression is, in effect, a functional call, multiple stacks and multiple stack buffers are implemented in the hardware of the LISP machine. A stack on the 3600 (LISP machine) is managed by the processor hardware, which maintains various pointers to the stack. Stack-buffer manipulations such as push or pop are carried out by the processor and occur in one machine cycle. Fast function calling is critical to the performance of processor-bound programs. (Symbolics, 1984)

subgoal

A hierarchy of goals structured to satisfice another goal in a hierarchy of goals. Inference engines in expert system software use backward chaining systems where unmet goals are decomposed into subgoals. Problem solving proceeds to solve the simpler subgoals in an attempt to satisfice the goal.

symbolic processing

Adapting computer hardware to manipulate symbols.

symbolic reasoning
A type of reasoning similar to human thought processes. For example, the cognitive process of concept formation uses symbolic reasoning. A concept can be described by identifying the characteristics of the objects or events to which is applies. It can also specify measurable relations between the characteristics or features of the concept. Symbolic reasoning involves problem solving based on the application of strategies and heuristics (rules of thumb) used to manipulate the symbolic representations of the problem.

syntactic
Dealing with form or structure in AI software.

thinking
I. Thinking is a complex multifacted process. It is essentially internal (and possibly nonbehavioral), involving symbolic representations of events and objects not immediately present, but is initiated by some external event (stimulus). Its function is to generate and to control overt behavior. (Bourne, Ekstrand, and Dominowski, 1971, p 4) II. 1. Any process or activity not predominantly perceptual by which one apprehends an object or some aspect of an object or situation. This implies judging, abstracting, conceiving, reasoning, and (in a somewhat extended sense) imagining, remembering, and anticipating are forms of thinking. Although thinking is thus negatively defined by reference to perceiving, the two processes are not antagonistic but supplemental. Either may merely predominate in any given cognitive process. 2. Problem solving that involves primarily ideas rather than perceiving and overt manipulation. 3. Meditating or reflecting upon a problem in order to understand the relationships involved...(English and English, 1958, p. 553)

toolkit
Computer programs that are used to build AI software. Toolkit software typically contains an inference engine, knowledge acquisition aids, and user/programer interfaces. Toolkit software evolved from expert systems where the knowledge base has been removed. Toolkits are recommended for scoping the construction of a large, high resolution expert system.

toy problem
A simplified adaptation of AI to solve complex problems for demonstration purposes. Toy problems often hide the difficulty in adapting AI techniques to solve real-world problems.

uncertainty

A value that cannot be determined in an expert system program. A representation of uncertainty is used to describe a belief (or disbelief) in symbolic or numeric data. For example, the numerical representation of scalars on an interval (i.e., the Bayesian approach to statistics), or in logic a distribution on a universe of discourse (i.e., the extension of necessity and possibility), or as points in space (evidence space). AI programing deals with uncertainty with such techniques as fuzzy reasoning.

user interface

The component of an expert system that allows bidirectional linkages between the software, hardware, programer, and user.

VLSI

Very large scale integration. The process of producing integrated circuits containing many thousands of electronic devices.

ARTIFICIAL INTELLIGENCE STUDY (AIS)	**STUDY SUMMARY CAA-RP-87-1**

THE REASON FOR PERFORMING THE STUDY was to gain a preliminary overview of adapting artificial intelligence (AI) and expert system technology to the theater-level for modeling and for the CAA study process.

THE PRINCIPAL ACCOMPLISHMENTS of the study are:

(1) AI software, hardware, and toolkits were surveyed for their applicability to CAA's study and analysis processes.

(2) AI techniques and methodologies were surveyed to determine their application to CAA study process.

(3) A small demonstration project was developed using the Command and Control (C^2) logic from CAA's FORCEM simulation model. The code was developed using the Knowledge Engineering Enviornment (KEE) toolkit on a Symbolics 3670 Lisp Machine.

THE MAIN ASSUMPTIONS OF THE STUDY are:

(1) AI/expert system technology may be applicable to military modeling.

(2) Rule-based solutions may be implemented to support computer simulation models used at CAA.

THE SPONSOR was the Director, US Army Concepts Analysis Agency.

THE STUDY EFFORT was directed by Dr. Richard B. Modjeski, Advanced Research Projects Office of the US Army Concepts Analysis Agency.

COMMENTS AND QUESTIONS may be addressed to the Director, US Army Concepts Analysis Agency, ATTN: CSCA-RSR, 8120 Woodmont Avenue, Bethesda, MD 20814-2797.

ARTIFICIAL INTELLIGENCE STUDY (AIS)	STUDY SUMMARY CAA-RP-87-1

THE REASON FOR PERFORMING THE STUDY was to gain a preliminary overview of adapting artificial intelligence (AI) and expert system technology to the theater-level for modeling and for the CAA study process.

THE PRINCIPAL ACCOMPLISHMENTS of the study are:

(1) AI software, hardware, and toolkits were surveyed for their applicability to CAA's study and analysis processes.

(2) AI techniques and methodologies were surveyed to determine their application to CAA study process.

(3) A small demonstration project was developed using the Command and Control (C^2) logic from CAA's FORCEM simulation model. The code was developed using the Knowledge Engineering Enviornment (KEE) toolkit on a Symbolics 3670 Lisp Machine.

THE MAIN ASSUMPTIONS OF THE STUDY are:

(1) AI/expert system technology may be applicable to military modeling.

(2) Rule-based solutions may be implemented to support computer simulation models used at CAA.

THE SPONSOR was the Director, US Army Concepts Analysis Agency.

THE STUDY EFFORT was directed by Dr. Richard B. Modjeski, Advanced Research Projects Office of the US Army Concepts Analysis Agency.

COMMENTS AND QUESTIONS may be addressed to the Director, US Army Concepts Analysis Agency, ATTN: CSCA-RSR, 8120 Woodmont Avenue, Bethesda, MD 20814-2797.

CAA	ARTIFICIAL INTELLIGENCE STUDY (AIS)	STUDY SUMMARY CAA-RP-87-1

THE REASON FOR PERFORMING THE STUDY was to gain a preliminary overview of adapting artificial intelligence (AI) and expert system technology to the theater-level for modeling and for the CAA study process.

THE PRINCIPAL ACCOMPLISHMENTS of the study are:

(1) AI software, hardware, and toolkits were surveyed for their applicability to CAA's study and analysis processes.

(2) AI techniques and methodologies were surveyed to determine their application to CAA study process.

(3) A small demonstration project was developed using the Command and Control (C^2) logic from CAA's FORCEM simulation model. The code was developed using the Knowledge Engineering Enviornment (KEE) toolkit on a Symbolics 3670 Lisp Machine.

THE MAIN ASSUMPTIONS OF THE STUDY are:

(1) AI/expert system technology may be applicable to military modeling.

(2) Rule-based solutions may be implemer·ed to support computer simulation models used at CAA.

THE SPONSOR was the Director, US Army Concepts Analysis Agency.

THE STUDY EFFORT was directed by Dr. Richard B. Modjeski, Advanced Research Projects Office of the US Army Concepts Analysis Agency.

COMMENTS AND QUESTIONS may be addressed to the Director, US Army Concepts Analysis Agency, ATTN: CSCA-RSR, 8120 Woodmont Avenue, Bethesda, MD 20814-2797.

END
6-87
DTIC